FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Plataforma para a Conceção, Simulação e Desenvolvimento de Sistemas Multiagente de Quadcopters

Rúben Filipe Delindro Veloso

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Prof. Dr. Rosaldo José Fernandes Rossetti

Co-orientador: Eng. Lúcio Sanchez Passos

Co-orientador: Eng. Zafeiris Kokkinogenis

17 de Fevereiro de 2014

Plataforma para a Conceção, Simulação e Desenvolvimento de Sistemas Multiagente de Quadcopters

Rúben Filipe Delindro Veloso

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Prof. Dr. Carlos Manuel Milheiro de Oliveira Pinto Soares
Arguente: Prof. Dr. Artur José Carneiro Pereira
Vogal: Prof. Dr. Rosaldo José Fernandes Rossetti

17 de Fevereiro de 2014

Resumo

A investigação na área dos Veículos Aéreos Não Tripulados (VANTs), mais concretamente sobre os *quadcopters*, têm vindo a ser aprofundada cada vez mais nos últimos anos, uma vez que este tipo de veículos têm uma grande capacidade de manobra e de estabilidade, sendo procurados para executar as mais variadas tarefas. Através das ferramentas de simulação de veículos autónomos, novas metodologias e algoritmos de controlo foram sendo desenvolvidos. No entanto, no que diz respeito à conceção, desenvolvimento e validação dos veículos, estes simuladores ainda não reúnem as ferramentas necessárias.

Assim, a proposta desta dissertação consiste em especificar, implementar e testar uma plataforma de simulação para ajudar no desenho e desenvolvimento de VANTs. Este projeto considera apenas as aplicações no domínio dos *quadcopters* em cenários de aplicação civil. A plataforma de simulação funciona como tal para suportar as tomadas de decisões relativas à conceção e implementação de agentes VANT. Serve também como ferramenta para desenvolver e experimentar novas estratégias de controlo, cooperação e coordenação entre agentes autónomos.

Após uma análise do estado da arte com base nas áreas de contexto em que o projeto se insere, foram apresentados os requisitos, proposta uma arquitetura e definidos os pressupostos que vão limitar o foco da abordagem a seguir. Do ponto de vista da sua definição, encontram-se cinco objetivos principais começando pela especificação e implementação da meta-arquitetura e do ambiente virtual dos agentes VANT. E também pela própria implementação dos sensores e respetivos mecanismos para a simulação simbiótica, isto é, a ligação entre um *quadcopter* real e um virtual.

Através da implementação e integração das várias tecnologias, como a simulação simbiótica, a co-simulação e os sistemas multiagentes, foi possível criar uma plataforma que permite reunir os pontos mais fortes de cada tipo de sistema por forma a otimizar os resultados nas tomadas de decisões dos agentes VANTs quer no ambiente virtual, como no real, através da ligação simbiótica.

Por forma a se avaliar a plataforma de simulação, foram executados alguns testes de funcionalidade assim como de desempenho e foram analisados e discutidos os resultados obtidos tendo em conta os objetivos referidos. Por fim apresentam-se possíveis melhoramentos e trabalhos futuros, dado que a integração das múltiplas tecnologias em conjunto vai abrir portas para várias áreas de investigação e também de testes de novas estratégias e algoritmos que venham a ser desenvolvidas.

Palavras-chave: Simulação Simbiótica, Co-Simulação, Veículos Aéreos Não Tripulados, *Quadcopters*, Sistemas Multiagente

Abstract

A Platform for the Design, Simulation and Development of Quadcopter Multi-Agent Systems

The aim of the project described in this dissertation proposal is to specify, implement and test a simulation platform to assist in the design and development of UAVs. This project only considers applications in the field of application scenarios in quadcopters civil.

The platform as a whole, should allow quadcopters autonomous and intelligent to be able to perform simple tasks, such as maneuvering training, or running missions and plans without recourse to human decisions.

The platform acts as a simulator for the optimization of decision making simulated by multi-agent systems, that can perform their tasks avoiding collisions and obstacles in an effective way by collecting data from the sensors and also by communicating with each other.

Since agents will represent quadcopters real, they must implement strategies for coordination, cooperation and collaboration for the various possible configurations of quadcopters. These coordinations among quadcopters will be implemented with the system used by aircraft that is called TCAS collision avoidance system. It will also allow symbiotic simulation by connecting a real *quadcopter* to a virtual one through the use of the XBee modules. These modules allow communication by radio waves over long distances using a few energy resources and use a communication protocol that allows the exchange of information called UAVTalk.

This project is at an early stage of their development, so that is will be the first version of the simulator to be developed relying instead only some prototypes made during the study of the available technologies.

From the point of view of definition, there are five major goals starting from the specification and implementation of the architecture for the meta-quadcopters, and the physical model of the environment. Passing by the very implementation of the simulator and then immediately the implementation of the communication protocol between agents and finally connecting the real *quadcopter* with the virtual one.

Achieving all of these presented goals, the project will meet all the requirements for the development of simulation platform for intelligent and autonomous quadcopters desired.

Keywords: Symbiotic Simulation, Co-Simulation, Unmanned Aerial Vehicles, Quadcopters, Multi-Agent Systems

Agradecimentos

Este espaço está reservado apenas para as pessoas que eu considerei mais importantes e que fizeram parte deste meu percurso académico na Faculdade de Engenharia da Universidade do Porto (FEUP) e para o qual o seu contributo foi imprescindível.

Para não incorrer em qualquer tipo de injustiças, agradeço já desde antemão a todos os que se cruzaram comigo sem qualquer exceção e que de algum modo foram essenciais para que eu tenha concluído a minha dissertação com êxito.

Esta só foi possível devido ao contributo e à colaboração prestada pelo meu orientador Prof. Dr. Rosaldo José Fernandes Rossetti e dos co-orientadores Eng. Lúcio Sanchez Passos e Eng. Zafeiris Kokkinogenis aos quais quero direcionar algumas palavras de apreço e de reconhecimento.

A todos eles pelo interesse evidenciado e pela disponibilidade manifestada para orientar na definição do meu objetivo de estudo, pela exigência de método e rigor impostos, por todos os esclarecimentos, opiniões e sugestões. Enfim por toda a acessibilidade e confiança que sempre demonstraram.

Gostaria de agradecer também ao Laboratório de Inteligência Artificial e Ciências da Computação (LIACC) por me ter recebido no seu grupo desde o início.

Por último e não menos importante, deixo aqui um agradecimento muito especial aos meus pais, irmão e amigos pelo apoio incondicional ao longo deste meu percurso académico sendo todos eles modelos de coragem e incentivo. Estiveram sempre presentes e continuam a estar em todos os momentos mais importantes da minha vida e por isso direciono a todos eles o meu profundo agradecimento.

Rúben Veloso

"Education is the most powerful weapon which you can use to change the world."

Nelson Mandela

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Abreviaturas e Símbolos

3D	3-Dimensões
ADV	Aterragem e Descolagem Vertical
AFA	Administração Federal de Aviação
CArtAgO	*Common ARTifact infrastructure for AGents Open environments*
CUDA	*Compute Unified Device Architecture*
ECS	Estação de controlo no solo
FEUP	Faculdade de Engenharia da Universidade do Porto
IDMEC	Instituto de Engenharia Mecânica
IA	Inteligência Artificial
IAD	Inteligência Artificial Distribuída
JADE	*Java Agent DEvelopment Framework*
JDK	*Java Development Kit*
jME	*Java Monkey Engine*
JVM	*Java Virtual Machine*
LIACC	Laboratório de Inteligência Artificial e Ciências da Computação
PFOA	Potencialidades, Fraquezas, Oportunidades e Ameaças
PI	Proporcional Integral
PID	Proporcional Integral Derivativo
RPD	Resolução de Problemas Distribuídos
SACT	Sistema Anti Colisão de Tráfego
SCF	Sistemas Ciber-Físicos
SCSS	Sistema de Controlo de Simulação Simbiótica
SDK	*Software Development Kit*
SMA	Sistemas Multiagente
SPSS	Sistema de Previsão de Simulação Simbiótica
SSADS	Sistema de Detecção de Anomalias de Simulação Simbiótica
SSDSS	Sistema de Suporte à Decisão de Simulação Simbiótica
SVMSS	Sistema de Validação do Modelo de Simulação Simbiótica
SBMA	Simulação baseada em Multiagentes
UML	*Unified Modeling Language*
VANT	Veículo Aéreo Não Tripulado
VTNT	Veículo Terrestre Não Tripulado

Capítulo 1

Introdução

Este capítulo apresenta a contextualização e motivação para o desenvolvimento da plataforma de simulação de *quadcopters*, assim como os objetivos que se pretendem alcançar desde a implementação da autonomia até à ligação do mundo virtual (ou seja, simulado) com o mundo real. Por último é apresentada a estrutura deste documento.

1.1 Contexto e Motivação

Durante os últimos anos, o número de aplicações de simulação com os veículos aéreos não tripulados (VANTs) evoluiu rapidamente no surgimento de plataformas com um elevado nível de autonomia. Tanto em aplicações militares como em civis, os VANTs são utilizados para executar tarefas que ocorrem em locais perigosos e/ou inacessíveis para os seres humanos. Embora estes sistemas incorporem diferentes graus de independência e autonomia, todos eles ainda necessitam de supervisão humana e das respetivas tomadas de decisão, que se encontram implementadas na maioria das plataformas analisadas.

O âmbito desta dissertação é um subconjunto dos veículos autónomos aéreos, os *quadcopters*, que abrangem várias áreas de aplicação. Estes VANTs têm uma grande capacidade de manobra e de estabilidade, pelo que são utilizados para executar as mais variadas tarefas. Este projeto surge de uma cooperação do Laboratório de Inteligência Artificial e Ciências da Computação (LIACC) com o Instituto de Engenharia Mecânica (IDMEC), ambos pertencentes à Faculdade de Engenharia da Universidade do Porto (FEUP).

Para o desenvolvimento da plataforma foram abordadas as áreas dos Sistemas Multiagente (SMA) e da Simulação. Esta última divide-se em dois tipos: a primeira abordagem para a simulação baseada em *quadcopters* virtuais e a segunda abordagem, a simulação simbiótica, que permite a ligação do *quadcopter* virtual com o real.

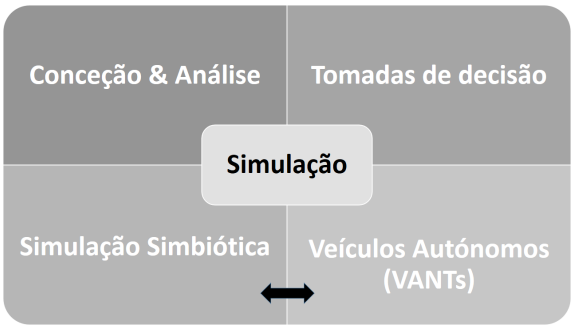


Figura 1.1: Áreas de Contexto do Projeto

Ambas as abordagens da simulação dos *quadcopters* autónomos serão tratadas e serão implementadas as tomadas de decisão e a conceção/análise dos modelos no qual se pretendem implementar estratégias de cooperação, coordenação e colaboração. O esquema do contexto do projeto apresenta-se na Figura 1.1 e como se pode observar a simulação vai agregar os quatro tópicos previamente referidos. Também é de destacar que a simulação simbiótica vai permitir a ligação com os VANTs reais evidenciado pela seta de duplo sentido, que vai ser um dos objetivos a alcançar no desenvolvimento da plataforma.

A motivação para a elaboração desta dissertação adveio da perspetiva futura de implementação dos veículos aéreos autónomos num ambiente real, onde todos os utilizadores possam desenvolver os seus *quadcopters* e testá-los em ambientes simulados sem a necessidade de adquirir os equipamentos reais. Reduzindo assim desta maneira os riscos inerentes à danificação do equipamento real para valores bastante reduzidos, uma vez que podem ser equipamentos bastante dispendiosos e únicos.

Devido às enormes quantidades de modelos e configurações possíveis, surge uma motivação maior para o desenvolvimento e implementação de várias arquiteturas, assim como estratégias de coordenação e cooperação entre os *quadcopters*. A especificação de protocolos de comunicação também vai permitir ajudar a estabelecer uma melhor interação entre os *quadcopters* reais e virtuais.

É um projeto bastante complexo e aliciante pelo que a existência da utilização destes equipamentos trouxe também uma motivação adicional. De referir também que a aplicação desta plataforma em várias áreas pode trazer vantagens competitivas em relação às plataformas já existentes.

1.2 Problema de Investigação: Definição e Objetivo

A simulação de sistemas complexos é essencial para o seu desenvolvimento e serve para evitar a ocorrência de falhas quando testadas num cenário real. Isso é especialmente verdade para os veículos aéreos não tripulados pois a sua falha normalmente significa danos permanentes nos

equipamentos ou mesmo a sua destruição completa. Além disso, sistemas autónomos e cooperativos podem interagir e consequentemente resultar em cadência de eventos não-determinísticos. No nosso caso, a cooperação emergente de um grupo de *quadcopters* é um exemplo desse não-determinismo. Do ponto de vista de um único VANT, a simulação proporciona a possibilidade de analisar diferentes aspetos, por exemplo, em termos físicos (estabilidade/dimensionamento) e em termos de capacidade de cognição (controlo de voo baseado em aprendizagem).

Existem alguns simuladores que permitem a integração de múltiplas instâncias de VANTs, apesar de todos estes terem uma capacidade limitada de simulação com múltiplos *quadcopters* em ambiente virtual e interação com o mundo real. Além disso, a maioria dos sistemas são implementados com um único VANT em vez de um conjunto deles. Por sua vez pode-se observar a falta de aplicações para modelação e simulação em 3-Dimensões (3D) de múltiplos *quadcopters* em ambiente virtual.

Outro grande problema identificado, consiste na capacidade de controlo de um utilizador humano visto que normalmente este interage ou controla um único VANT de cada vez. Presume-se que será devido ao custo associado ao equipamento e à grande complexidade do seu controlo. Para além dos custos associados aos equipamentos, existe a necessidade de realizar os testes em ambiente simulado para evitar a danificação do equipamento real.

Ainda se pretendem implementar estratégias de coordenação e controlo para aplicações civis em que vários *quadcopters* são usados para missões rápidas e de curtas distâncias como reconhecimento, recolha de análises, entre muitas outras tarefas de uma forma autónoma.

Após identificados todos estes problemas, o objetivo global que se pretende atingir nesta dissertação consiste em especificar, implementar e testar uma plataforma de simulação para ajudar na conceção e desenvolvimento de VANTs. Este projeto considera apenas as aplicações no domínio dos *quadcopters* em cenários de aplicação civil.

Estes *quadcopters* devem ter um certo nível de tomada de decisão por forma a serem autónomos a fim de conseguirem executar tarefas simples, como deslocações para uma determinada posição, ou executando missões e planos sem recurso a decisões humanas.

Dado o nível de complexidade e a quantidade de trabalho inerente às várias áreas que este projeto abrange, desde os sistemas multiagente até a simulação simbiótica, este projeto encontra-se dividido em cinco objetivos principais que se encontram listados de seguida:

- **Especificar e implementar a meta-arquitetura dos Agentes VANT** – Este primeiro objetivo permite modelar o *quadcopter*, que vai ser especificado por quatro componentes que são: o modelo elétrico, a dimensão física, os sensores e as restrições computacionais. Este modelo deve ser capaz de representar um *quadcopter* real num ambiente simulado.

- **Especificar e implementar o ambiente virtual dos Agentes VANT** – O objetivo número dois consiste na especificação do modelo físico do ambiente no qual se encontram inseridos os *quadcopters*. Este ambiente pode ser estático ou dinâmico, caso se considerem os vários *quadcopters* em movimento no mesmo ambiente.

- **Implementar um simulador** – O objetivo número três consiste na implementação do simulador 3D para integração dos modelos de *quadcopters* e dos cenários de ambiente predefinidos. Este simulador deve permitir reproduzir um conjunto de situações com os SMA para que estes possam executar as suas tomadas de decisão e manter o processamento mais pesado à parte do *quadcopter*. Os *quadcopters* devem poder interagir e comunicar no mundo virtual ou no mundo virtual e real simultaneamente.

- **Especificar e implementar um protocolo de comunicação** – O objetivo número quatro consiste na especificação e implementação de um protocolo de comunicação entre múltiplos *quadcopters* virtuais ou adaptar algum protocolo já existente. Isto vai permitir definir o domínio de conhecimento de cada um, para que todos tenham o mesmo entendimento de cada informação desde os seus movimentos, ações, negociações, discussão mútua, entre outras propriedades que vão permitir a coordenação, colaboração e cooperação entre si.

- **Especificar e implementar os mecanismos para a simulação simbiótica** – Por fim, temos o objetivo número cinco que conta com a implementação da conexão do mundo virtual com o real através dos módulos XBee[1]. Esta simbiose pode ser utilizada em vários sentidos: para recolha de dados e processamento no mundo virtual, para enviar ações para o mundo real, ou de um modo híbrido que permite a troca em simultâneo entre os dois mundos. Ou seja, para especificar e implementar os mecanismos apropriados para integrar o ambiente virtual simulado com o *quadcopter* real.

Ao se atingirem todos estes objetivos apresentados, estarão reunidos todos os requisitos para o desenvolvimento da plataforma de simulação de *quadcopters* autónomos.

1.3 Abordagem Metodológica e Contribuições Esperadas

Em relação à abordagem metodológica, esta tem que seguir um caminho lógico para toda a implementação da plataforma funcionar, através da especificação e modelação o mais rigorosa possível.

Para tal e de acordo com a ordem dos objetivos propostos, o primeiro passo a seguir é a modelação do *quadcopter*. Neste objetivo vão ser utilizadas todas as informações possíveis que se conheçam e que estejam presentes nos componentes do equipamento. Desta forma, vai-se conseguir criar um modelo genérico, que mediante uma configuração rápida dos seus componentes possa representar o equipamento que o utilizador pretende testar na simulação. Esta abordagem vai contribuir para uma melhor modelação dos *quadcopters* uma vez que as modelações nos simuladores existentes estão limitadas a um modelo único.

De seguida surge a modelação do ambiente físico uma vez que se pretende simular um *quadcopter* num ambiente virtual e é necessário criar infraestruturas virtuais que reúnam as condições para os testes que se pretendem simular. Criando-se alguns modelos genéricos, com efeitos como

[1] `http://www.digi.com/xbee/`

o vento a circular, chuva, ou outras situações que sejam esperadas de acordo com as tarefas a executar. Desta forma, pode-se permitir aos agentes executar os testes autonomamente e assim contribuírem para encontrarem as melhores formas de executarem as tarefas de acordo com as métricas definidas.

Em terceiro lugar, surge a implementação dos modelos especificados num simulador 3D, esta é a plataforma que vai permitir a configuração dos modelos e das respetivas definições para reproduzirem situações num número elevado de vezes, procurando as melhores otimizações para as execuções das tarefas. Esta plataforma vai permitir a integração de modelos configuráveis de *quadcopters* e a ligação entre o mundo real e virtual agregando assim um valor acrescentado em relação às plataformas já existentes e mais limitadas.

Após a implementação da base da plataforma de simulação, vai-se proceder à implementação dos protocolos de comunicação entre os *quadcopters*. Isto vai permitir a troca de informações entre VANTs e por sua vez a integração das estratégias de coordenação, colaboração e cooperação. Supondo que uma simulação vai ter um elevado número de *quadcopters* no ambiente e que todos estes implementam um Sistema Anti Colisão de Tráfego (SACT), conseguir-se-á que os *quadcopters* se cruzem nas suas trajetórias evitando as colisões eminentes em sistemas sem supervisão humana, contribuindo desta forma a uma otimização das trajetórias dos agentes autónomos.

Por último, surge a implementação da simulação simbiótica que vai permitir otimizar o *quadcopter* real através dos agentes do simulador, retirando o que cada mundo oferece de melhor, por exemplo recolhendo dados reais dos sensores e aplicando esses dados nos algoritmos do simulador para otimizar as tomadas de decisão, sendo esta a contribuição esperada através desta ligação simbiótica.

1.4 Estrutura da Dissertação

A dissertação é composta por cinco capítulos: No Capítulo 1 efetua-se a contextualização e a apresentação do problema de investigação assim como a abordagem metodológica a seguir.

No Capítulo 2 faz-se um levantamento dos fundamentos teóricos, mais concretamente sobre os Sistemas Multiagente, a história dos VANTs, Sistemas Ciber-Físicos e a Simulação. Por fim é apresentado o estado da arte, ou seja, são apresentados os trabalhos relacionados e uma discussão crítica sobre toda a revisão da literatura.

No Capítulo 3 são apresentados os requisitos e os pressupostos identificados para o projeto. No que diz respeito à arquitetura, é feita uma primeira abordagem geral e de seguida aprofunda-se mais em pormenor a implementação de cada requisito. Faz-se também uma discussão crítica de todo o capítulo.

No Capítulo 4 são apresentados os cenários de testes e validações dos resultados, através de métricas de funcionalidade e de desempenho.

Por fim no Capítulo 5 apresentam-se as considerações finais, seguido da análise de satisfação dos objetivos e trabalho futuro, concluindo com uma análise de todo o projeto.

No Apêndice A encontra-se o ambiente de desenvolvimento onde é descrito a seleção das ferramentas e respetivas configurações. No Apêndice B é apresentado o artigo desenvolvido durante a dissertação. Por último no Apêndice C são apresentados os diagramas *Unified Modeling Language* (UML) mais significativos.

Capítulo 2

Revisão Bibliográfica

Neste capítulo são apresentados os fundamentos teóricos dos Sistemas Multiagente, História dos VANTs, Sistemas Ciber-Físicos e da Simulação. Define-se toda a área de investigação sobre o projeto em si e permitindo refletir sobre como abordar a resolução do problema, de uma forma mais rápida sem reinventar ideias bem definidas e estruturadas. Também será apresentado o estado da arte fazendo uma análise a alguns projetos relacionados e por último uma discussão crítica sobre todo o capítulo.

2.1 Fundamentos Teóricos

Os fundamentos teóricos e o estado da arte são a parte científica do projeto, no qual se faz referência às investigações e aos trabalhos desenvolvidos pela comunidade sobre o assunto em estudo. Permite analisar o que já se encontra desenvolvido quer na teoria quer na prática tornando-se uma atividade bastante extensa e árdua por ser uma tarefa crítica no qual se implica muita reflexão.

2.1.1 Sistemas Multiagente

O conceito de programas autónomos surge na década de 90, relacionado com a área da Inteligência Artificial (IA) numa tentativa de criar programas que conseguissem imitar o comportamento humano. Após esse momento, passou a ser um tema de eleição por parte da comunidade de IA.

Inicialmente, os sistemas compostos por múltiplos agentes eram associados à Inteligência Artificial Distribuída (IAD), mais tarde vieram a ser divididos em dois campos distintos: a Resolução Distribuída de Problemas (RDP) e Sistemas Multiagente (SMA) [DR94]. O primeiro assume que cada entidade tem habilidades específicas e que todos os agentes dependem de uma execução coordenada de ações. Já os SMA "estão associados à coordenação de comportamento inteligente entre um grupo de (possivelmente pré-existente) agentes autónomos e inteligentes: estes coordenam

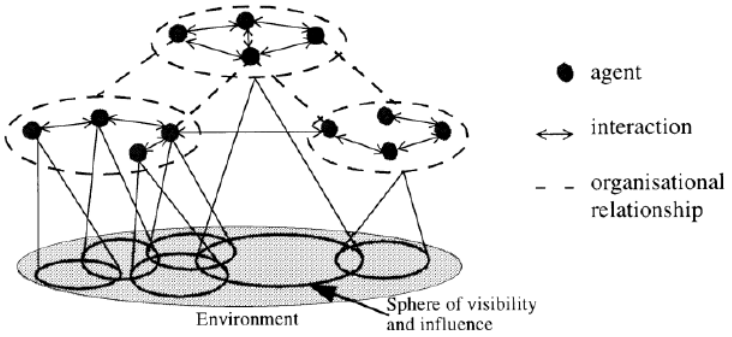


Figura 2.1: Perspetiva canónica de um sistema baseado em agentes [UW09]

seus conhecimentos, objetivos, habilidades e planos juntamente para executar ações e resolverem (possivelmente múltiplos e independentes) problemas" [Gas87].

Como pode ser observado na Figura 2.1, os SMA são compostos por agentes que têm uma esfera de visibilidade e influência sobre o ambiente. Além disso, eles estão organizados em grupos que podem ser formados através da identificação de objetivos comuns onde os agentes podem interagir de forma autónoma. De seguida, os principais conceitos identificados na figura serão mais bem descritos e discutidos.

2.1.1.1 Agentes Autónomos

O conceito de agente autónomo trouxe grandes discussões ao longo do tempo na comunidade científica. Ainda assim uma única definição nunca foi conseguida, pois cada área de aplicação envolvendo agentes, moldava esse conceito de acordo com as suas próprias necessidades. De uma forma geral, entende-se por agente autónomo, um agente único e independente da existência de outros agentes. Este deve possuir um conjunto de capacidades, objetivos e autonomia para atingir os seus propósitos [Vla07].

Várias definições sobre o conceito de agente podem ser encontradas em literatura [FG97, Flo98, TU10]. Por exemplo Russell and Norvig define um agente autónomo como uma entidade que tenha a capacidade de perceção do ambiente e a capacidade de agir nesse ambiente através de atuadores. Desta forma o agente poderá efetuar a melhor decisão para atingir os seus objetivos.

A definição que vai ser adotada nesta dissertação foi apresentada por Jennings [JSW98] em 1998. Este define um agente como um sistema de computação, com orientação/coordenação em um determinado ambiente e com a capacidade para executar ações flexíveis autonomamente em ordem para atingir os objetivos para o qual foi desenhado, ou seja, esta definição segue os seguintes conceitos fundamentais: autonomia, contextualização, reatividade e pró-atividade.

- **Autonomia** — Um agente opera sem a intervenção direta de humanos ou outros e tem controlo sobre as suas ações e estado interno;
- **Contextualização** — Os agentes pertencem a um ambiente e interagem com outros agentes via algum tipo de linguagem. Não é um agente isolado mas sim um agente da sociedade;

- **Reatividade** — Os agentes têm a perceção do seu ambiente e respondem em tempo útil;
- **Pró-atividade** — Os agentes exibem um comportamento orientado para com os seus objetivos através da tomada de iniciativa.

Além disso, os agentes são entidades que necessitam também de uma estrutura interna específica. Essa estrutura muitas vezes é relacionada com a de um objeto e existem algumas discussões sobre a relação entre agente e as abordagens orientadas a objetos [TU10]. A estrutura de um agente e, consequentemente, o seu funcionamento interno é o que se denomina por arquitetura do agente. Os agentes são divididos em três grandes grupos quando classificados pela sua arquitetura:

- **Agentes Reativos** — Estes são unidades de processamento simples que contêm perceção e têm a capacidade de reagir às mudanças do ambiente [Flo98]. Algumas das características que mais se destacam são: não existe um estado do ambiente, todo o seu conhecimento é regido de acordo com as regras definidas. O comportamento de cada agente é baseado apenas no estado do momento uma vez que não têm um histórico das suas ações;
- **Agentes Cognitivos** — Estes tipos de agentes são baseados em formas de raciocínio bastante mais complexas. Algumas das características que estes agentes apresentam são: tem uma representação do estado do ambiente que os rodeia, têm um histórico das suas ações (memória), possuem um sistema de perceção e de comunicação;
- **Agentes Híbridos** — Esta terceira abordagem junta os agentes reativos com os cognitivos. Apesar de serem duas soluções opostas, em conjunto permitem criar uma arquitetura mais flexível. Nesta arquitetura os agentes são compostos por módulos, que permitem trabalhar independentemente dos comportamentos reativos e cognitivos. O grande problema que surge nesta arquitetura é encontrar um mecanismo de controlo mais eficaz que assegure um bom balanço e coordenação entre os módulos [MF09].

2.1.1.2 Ambiente para SMA

O papel do ambiente em SMA é alvo de investigação há bastante tempo no domínio do comportamento dos agentes, que consideram a interação no ambiente como uma funcionalidade crucial para existir de certo modo um comportamento inteligente [Bro91]. A referência [OVDPFB03] afirma que um ambiente deve garantir todas as condições sobre as quais uma entidade (agente ou objeto) existe, ou seja, garante os princípios e processos que regulamentam e suportam uma população de entidades. Russell e Norvig [RN03] analisam os ambientes numa perspetiva orientada às tarefas que vão ser executadas, no qual estas tarefas são essenciais para os problemas que os agentes racionais vão tentar resolver.

Danny Weyns [WOO07] apresenta uma abordagem onde coloca o ambiente como uma abstração de primeira ordem nos SMA, isto em grande parte devido ao fato de muitos aspetos dos SMA não pertencerem aos agentes ou estarem integrados dentro deles. Os autores concluíram

que o ambiente é o candidato natural para encapsular todos esses aspetos ajudando na gestão da complexidade da aplicação [WSR+05].

Para uma melhor especificação do ambiente, são agrupados nas seguintes dimensões baseadas em [RN03]:

- **Observável vs. Parcialmente observável** — Quando um agente tem um acesso completo ao estado do ambiente em um determinado momento, então o ambiente é considerado complemente observável. Isto é, se os seus sensores conseguirem detetar todos os aspetos relevantes e conseguirem recolher todas as informações necessárias para a tomada de decisão. Se um agente não integrar sensores de infravermelhos ou sonares então o ambiente é apenas parcialmente observável;
- **Determinístico vs. Estocástico** — De um forma geral, um ambiente é determinístico caso qualquer ação do agente provoque um efeito determinado no ambiente, no caso do estocástico essa ação pode ter um resultado diferente do previamente esperado pelo agente;
- **Episódico vs. Sequencial** — Em ambientes sequenciais, uma ação pode reverter em efeitos nos futuros estados do ambiente, enquanto que nos episódicos os estados futuros não dependem de ações executadas no passado;
- **Estático vs. Dinâmico** — Se o ambiente estiver em constante alteração enquanto um agente se encontra a deliberar, então é considerado um ambiente dinâmico. Caso contrário, quando os estados do ambiente se encontram sempre numa mesma faixa de valores, esse é considerado um ambiente estático.
- **Discreto vs. Contínuo** — O ambiente discreto pode-se diferenciar ao nível da resolução, precisão e do detalhe durante a perceção das ações do agente. Em um ambiente contínuo, o estado do ambiente está em constante mudança e não se consegue ter o detalhe referido anteriormente;

Segundo o que foi descrito anteriormente existem vários tipos de ambiente que um agente pode integrar, o que torna a implementação e desenvolvimento de um SMA completo (em qualquer domínio) num processo bastante complexo. Pelo que pode ser utilizada uma abordagem tradicional relacionada com o domínio dos transportes para a integração da plataforma [PRG11]. Segundo o autor é apresentada uma proposta para uma nova metodologia que considera os serviços como um conjunto de agentes, ambientes e processos.

2.1.1.3 Características para SMA

Quando múltiplos agentes trabalham em um mesmo ambiente, diversas outras características devem ser observadas para uma melhor adequação do Sistema Multiagente à aplicação. O primeiro aspeto que se tem que ter em consideração são as características do ambiente sobre o qual os agentes trabalharão. Segundo Jennings [Jen96], as aplicações que tiram melhor proveito do paradigma de SMA são as: modulares, descentralizadas, dinâmicas, não-estruturadas e complexas.

A comunicação é crucial para a existência de um sistema multiagente [Vla07]. A interação entre os agentes são sob a forma de envio e receção de mensagens entre si, sendo estas meramente informativas, de questão, de compromisso, de proibição ou de ordenação. A interação é uma abordagem que se pode tornar bastante satisfatória na resolução de problemas complexos, pois os agentes trocam informações sobre os problemas ou resolvem-nos em grupos.

Através da interação, os agentes podem formar sociedades. Segundo [Dem91] existem quatro tipos de sociedades em SMA: coabitação, cooperação, colaboração e distribuição. A coabitação é formada por um conjunto de agentes autónomos que executam tarefas individuais, onde cada agente executa a sua função em ordem ao objetivo geral. Em modo de colaboração, os agentes regem-se pela execução das suas tarefas localmente, de modo a conseguirem atingir a meta global de todos os agentes e onde em conjunto possam executar cada um a sua tarefa, como contribuição para a resolução dos objetivos gerais. Só deste modo é que se torna possível resolver as metas globais de uma forma coletiva.

Para se conseguir adotar esses comportamentos, os agentes devem partilhar conhecimentos, objetivos, ou planos para resolverem o problema. Numa sociedade de agentes, a direção e os objetivos nem sempre são alinhados. Assim surge a competitividade nos ambientes onde por exemplo, os recursos sejam limitados ou simplesmente que os seus objetivos sejam maximizar a sua função. Existem situações onde a negociação é usada como um meio intermediário para ambos atingirem os seus objetivos.

Os sistemas baseados em agentes podem ser considerados homogéneos quando todos os agentes são do mesmo tipo, ou heterogéneos quando são agrupados em agentes de tipos diferentes. No caso dos SMA heterogéneos, surgem vários problemas que são apresentados numa discussão bastante detalhada por Peter Stone e Manuela Veloso [SV00], onde apresentam os problemas que surgem com a falha de comunicação no sistema.

Outra característica do SMA é o seu nível de abertura. Sistemas abertos conseguem lidar com grandes fluxos de informação diversificada e ainda explorar a concorrência maciça [JSW98, HI91]. Deste modo podem surgir lacunas no controlo global de todo o sistema, desde falhas de consistência do conhecimento global, falha nas metas ou objetivos partilhados pelos agentes, ou até mesmo na visão global do próprio sistema.

Com o objetivo de consolidar o conjunto de conceitos apresentados ao longo dessa subsecção, diferentes taxonomias foram propostas [FG97, Sán97, Bru91, PRK11, TU10]. Depois de analisadas, escolheu-se a taxonomia de Andreas Tolk e Adelinde M. Uhrmacher [TU10]. Esta pretende dar resposta aos problemas apresentados na simulação com agentes, nos sistemas de engenharia e também tem em conta o ambiente em que se inserem. A taxonomia é apresentada na Figura 2.2 sendo classificada entre: características dos agentes, espaço onde os agentes se situam e sociedade dos agentes.

2.1.2 História dos VANTs

Os primeiros VANTs surgem por volta do século XIII quando o primeiro foguete chinês foi lançado, é claro que estes VANTs não tinham nenhum controlo. Os VANTs só começaram a ser

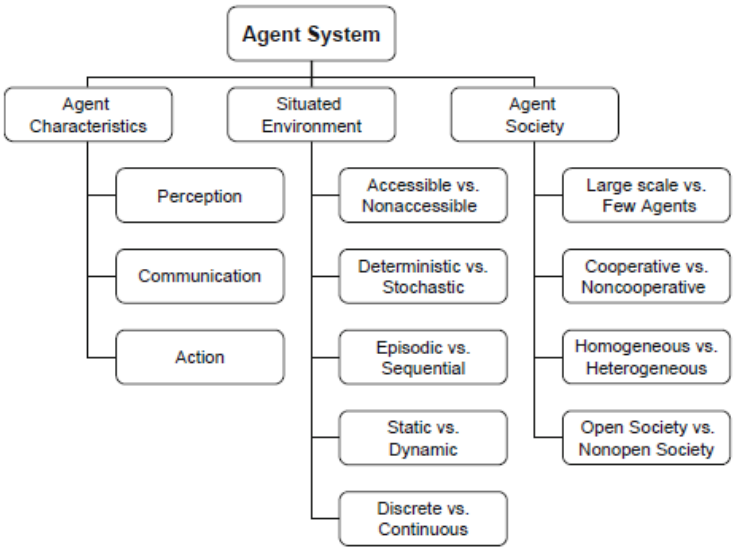


Figura 2.2: Taxonomia de Agentes [TU10]

reconhecidos como sistemas após a 1ª Guerra Mundial. Estes sistemas seriam compostos por três componentes: o veículo, a estação de controlo no solo (ECS) e o operador. Durante os últimos trinta anos, eles evoluíram para máquinas de grande capacidade e com várias funcionalidades. São também utilizados pelas forças militares em todo o mundo principalmente para vigilância e efeitos de aquisição de dados.

A classificação europeia dos VANTs identifica cinco categorias principais apresentadas na Tabela 2.1. Nas três primeiras podem-se encontrar os VANTs conhecidos pela sua capacidade de aterragem e descolagem vertical (ADV) bem como a pairar sobre o chão. Esta é a área onde se inserem os *quadcopters* que vão ser utilizados nesta dissertação. No entanto, surgiu também a necessidade de implementar estes produtos na indústria para aplicações civis, ajustando estes veículos para as necessidades mais particulares dos consumidores. Ainda assim, a introdução

Tabela 2.1: Classificação Europeia dos VANTs [SA01]

Classificação	**Alcance**	**Descrição**
Proximidade	<25Km	Veículos extremamente leves que podem ser lançados à mão.
Curto alcance	25..100Km	Veículos desenhados para operar em áreas limitadas.
Médio alcance	100..200Km	Veículos mais avançados aerodinamicamente com melhores sistemas de controlo para uma melhor eficácia operacional.
Longo alcance	200..500Km	Como na categoria anterior conseguem executar missões mais complexas, contam com ligações de satélite ou outro meio de comunicação que garante a ligação com o ECS.
Resistência	>500Km ou >20H	Categoria mais sofisticada de VANTs, distingue-se dos restantes pelas dimensões e capacidades superiores de tempo de voo.

Tabela 2.2: Aplicações Civis dos VANTs

Áreas	**Aplicações**
Vigilância	Patrulha fronteiriça e florestal.
Procura e Salvamento	Procura de sobreviventes de acidentes.
Industriais	Manufaturação, controlo de processos, telecomunicações, controlo de tráfego aéreo, sistemas de transporte.
Comerciais	Gestão de Informação, Comércio eletrónico.
Entretenimento	Jogos, Cinema.
Médicas	Monitorização de pacientes, Cuidados de Saúde.

desta tecnologia num mercado comercial é um problema multifacetado, sendo que existem ainda lacunas na resolução desses problemas [FGI09].

Várias são as áreas de aplicação que foram sendo exploradas ao longo dos tempos e algumas das possíveis aplicações deste tipo de sistemas são descritas na Tabela 2.2. Esta reúne alguns dos dados de estudo de Jennings [JSW98] e Zak Sarris [SA01] organizados por áreas de aplicação.

Os VANTs de vigilância surgiram inicialmente para fins militares, usados para sobrevoar as fronteiras ou outras situações de combate. Com a evolução das tecnologias, estes equipamentos foram introduzidos em aplicações industriais, mais recentemente, têm vindo a ser usados pelas forças de segurança pública para o controlo e vigilância das populações em grandes eventos. Em 2012, grandes investimentos foram feitos na aquisição de VANTs para a vigilância dos Jogos Olímpicos em Londres e para 2014 os mesmos vão ser utilizados no Mundial de Futebol do Brasil.

Além dos VANTs de vigilância, surgiram muitos outros projetos que utilizam estes veículos para a execução das mais variadas tarefas. Por exemplo, uma das aplicações é para a gestão e controlo de plantações, denominada de agricultura de precisão onde colocam VANTs equipados com câmaras especiais para recolher dados dos terrenos. Através da análise desses dados com algoritmos específicos, para cada situação podem-se aplicar os melhores tratamentos de acordo com as necessidades das plantações [TSLGDCPB13].

Outra aplicação dos VANTs de pequenas dimensões (como os *quadcopters*) é na área do cinema e da televisão onde são utilizados para a recolha de imagens obtidas através dos melhores planos aéreos, visto que se consegue uma melhor estabilidade com os mesmos.

2.1.3 Sistemas Ciber-Físicos

Os Sistemas Ciber-Físicos (SCF) são uma área que tem sido alvo de muita investigação por parte da comunidade, mas ainda é um assunto que se encontra em aberto. Mas o que são ao certo os SCF? Sistemas Ciber-Físicos são uma nova geração de redes conectadas de um conjunto de sistemas cibernéticos com sistemas físicos, que são controlados por regras definidas [TGP$^+$08].

Nesta dissertação, os agentes são considerados sistemas cibernéticos, sendo os *quadcopters* e os seus atuadores os sistemas físicos. O controlo incorporado com os SCF através dos *feedbacks* conseguem efetuar as tomadas de decisão mais otimizadas para cada situação. O impacto da informação dinâmica dos componentes físicos podem trazer um impacto negativo na componente

computacional devido à quantidade de informação que deve ser tratada. Esta informação deve ser tratada de uma forma modular dividindo os dados em blocos por forma a analisar-se somente o essencial [WBKS02].

Este tipo de sistemas devem integrar requisitos como a pontualidade, a precisão, a tolerância às falhas, a segurança, a escalabilidade e a autonomia. Ainda se precisa investir bastante no desenvolvimento desta tecnologia, uma vez que ainda existem problemas a serem resolvidos tais como, conseguir introduzir nos componentes físicos os requisitos de segurança e precisão. Além disso, os componentes físicos são diferentes dos componentes de software orientados a objetos [Lee08].

2.1.4 Simulação

A simulação é a única forma de desenhar, testar, analisar e estudar tanto a parte teórica como a parte real de um sistema computacional para os mais variados propósitos. Os agentes têm tido um papel essencial neste tipo de sistemas com a implementação de certas arquiteturas que ajudam na implementação dos protocolos de comunicação e coordenação de um sistema [TU10].

Na realidade com base nesta definição, a simulação pode ser considerada uma ferramenta computacional na qual deve obter um total entendimento de todo o sistema real, assim como também deve permitir o desenvolvimento de um sistema real funcional. As simulações servem assim para dar suporte em vários cenários, com o objetivo de treinar ou para ensinar agentes. É utilizada para simular os comportamentos e os processos de decisões individuais de pequenos grupos ou mesmo de populações inteiras, assim como os padrões de comunicação e cooperação entre si.

Numa perspetiva geral a validação dos sistemas baseados em simulação devem ser verificados de acordo com a implementação dos requisitos especificados. Para esta validação devem ser executados no mínimo um conjunto de testes críticos, por forma a garantir a fiabilidade do sistema e por conseguinte prevenir a danificação dos equipamentos reais.

2.1.4.1 Sinergias de Simulação

A sinergia é definida como um efeito ativo e retroativo das ações coordenadas de vários subsistemas na realização de uma tarefa complexa. Ou seja, é impossibilidade de se prever a partir da análise dos componentes de uma ação, o resultado atingido. Quando um conjunto de elementos realiza uma ação e analisando-se cada um desses elementos em separado é impossível prever o resultado obtido então não é considerada uma sinergia, só quando se reúnem as capacidades de cada elemento é que existe uma sinergia. Na Figura 2.3 estão representados os tipos de simulação com a interação de agentes.

Ören et al. [Ör02] identificam duas outras categorias possíveis: simulação autónoma e simulação embutida. A simulação autónoma é descrita como um programa que corre independentemente do sistema de interesse e encontra-se dividido em cinco tipos de utilização:

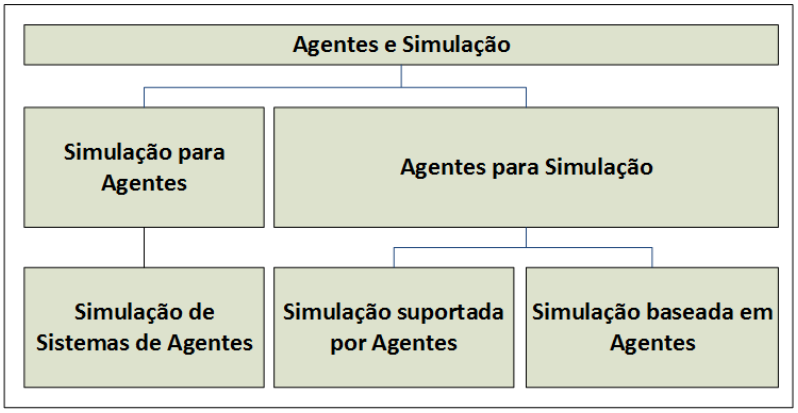


Figura 2.3: Tipos de Simulação direcionados a Agentes

- **Tomadas de decisão**, por exemplo, para a previsão de comportamentos, avaliação de alternativas, planeamento, *etc*.
- **Aprendizagem para melhoria das tomadas de decisão**, também conhecida como simulação construtiva.
- **Aprendizagem para melhoria das reações**, também utilizado nos simuladores virtuais comuns.
- **Ensino e Compreensão**, por exemplo, utilizado em ambientes académicos ou científicos.
- **Entretenimento**, por exemplo, utilizado em jogos de simulação, animações de sistemas dinâmicos.

No caso da simulação embutida, o programa de simulação corre em conjunto com o sistema físico e é dividido em dois objetivos:

- **Enriquecimento das tarefas do sistema real**, onde se processam os dados concorrentemente como, por exemplo, para o uso em diagnósticos em tempo real.
- **Suporte das tarefas do sistema real**, onde se processam os dados alternadamente para fornecerem resultados preditivos (para o uso de simulações em paralelo enquanto o sistema real está em execução).

Nas sinergias dos agentes e simulação são consideradas duas categorias: a primeira é a simulação para os agentes e a segunda são os agentes para a simulação que consiste, na simulação suportada por agentes e na simulação baseada em agentes [GAÖ01].

- **Simulação suportada por agentes** — É o uso da tecnologia de agentes que fornece suporte às atividades de simulação na modelação e na simulação de ambientes virtuais. Neste tipo de simulação, normalmente os agentes são utilizados para dar suporte às interfaces para o

processamento de elementos da simulação e ainda para tirar partido das funcionalidades cognitivas do sistema simulado como aprendizagem;

- **Simulação baseado em agentes** — É o uso da tecnologia de agentes para gerar modelos de comportamento ou para monitorizar a criação dos modelos de comportamento. É bastante útil para experiências complexas e para o processamento de todo o conhecimento obtido no planeamento, decisões e raciocínios.

A simulação suportada por agentes pode ser aplicada na plataforma que se pretende implementar uma vez que o sistema físico pode tirar vantagens diretamente da aprendizagem realizada no sistema simulado.

2.1.4.2 SMA e Simulação

Vários trabalhos têm vindo a ser tratados na relação entre os SMA e a simulação, sendo necessário destacar duas perspetivas: a modelação e simulação dentro dos próprios SMA e uma segunda perspetiva sobre os SMA em relação à modelação e simulação, ou seja utilizando os SMA numa perspetiva mais separada da simulação. No caso da primeira perspetiva referida, esta é usada como um meio para implementar as arquiteturas SMA, sejam quais forem os seus objetivos. Vários são os trabalhos que têm vindo a ser desenvolvidos, alguns deles são [MF09]:

- **The CNET Simulator** — É um sistema que especifica a resolução de problemas de comunicação e controlo para alocação de tarefas em um conjunto de nós distribuídos [Smi80];
- **The DVMT Project** — É um projeto de testes de monitorização de veículos distribuídos. Permitia aos investigadores explorar falhas de conceção em sistemas distribuídos para resolução de problemas [LC88];
- **MACE: Toward Modern Generic MAS Platform** — É considerada uma das primeiras plataformas genéricas dos SMA [GK02].

O objetivo desta dissertação é desenvolver uma plataforma de simulação capaz de desenhar, simular e desenvolver SMA de *quadcopters*. Esta simulação dos VANTs pode ser dividida em simulação virtual ou simulação simbiótica de agentes reais com virtuais como pode ser observado na Figura 2.4.

No que diz respeito à simulação nos VANTs, devem ser consideradas três componentes na sua modelação: a modelação dos agentes, a modelação do ambiente e a modelação da ligação entre os agentes e o ambiente. O primeiro componente encontra-se relacionado com o comportamento baseado na escolha de uma arquitetura e encontra-se discutido posteriormente na Secção 3.4. O segundo componente está relacionado com a definição do espaço virtual onde os agentes vão interagir, o tema do ambiente em SMA também já foi anteriormente referido na Secção 2.1.1.2.

Do ponto de vista da simulação, esta oferece suporte para teste de agentes individuais, grupos de agentes, geração de casos de teste sistemáticos, um melhor entendimento do ambiente onde o

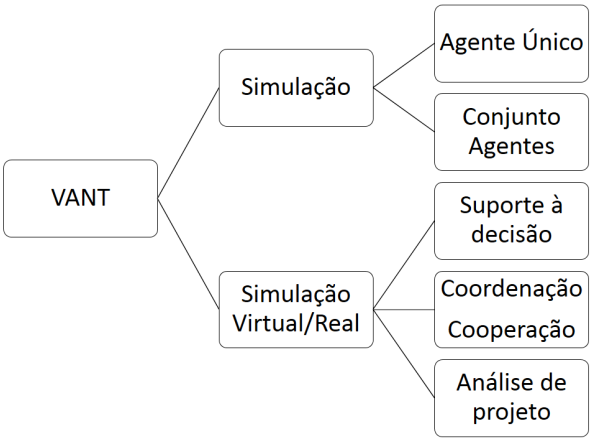


Figura 2.4: Simulação baseada em Multiagentes

agente se encontra. Por fim, os agentes e a simulação oferecem aos sistemas adaptabilidade, flexibilidade e robustez em ambientes dinâmicos e imprevisíveis. Os agentes reagem às oportunidades de atingir os seus objetivos específicos. Os agentes podem explorar a partilha de conhecimento dentro de uma organização de agentes.

Uma das muitas funcionalidades procuradas neste tipo de sistemas é o suporte de decisões analisando e avaliando a complexidade da dinâmica dos cenários através das simulações. Uma das implicações é a necessidade de executar mais rápido do que em tempo real, sendo este um problema que varia conforme a complexidade do sistema. Caso o sistema seja constituído unicamente por tarefas simples, não devem existir problemas de atraso no processamento exceto se for necessário implementar enormes quantidades de agentes a simular essas tarefas. O outro caso é quando é exigida uma enorme quantidade de processamento nos cálculos das tarefas de cada agente.

Os propósitos da utilização da modelação e simulação são: executar experiências para as tomadas de decisão para se obter melhor entendimento e aprendizagem [Ör10]. Garantem ainda as condições para efetuar o treino dos agentes em relação às deslocações, comunicações e tomadas de decisão. Por último, podem ainda ser utilizados para imitação de comportamentos para se obter uma melhor análise caso seja necessário.

2.1.4.3 Simulação Simbiótica

Simulação simbiótica é um paradigma no qual os sistemas de simulação e os sistemas físicos se encontram fortemente relacionados um com o outro [ATCL09a]. Surgiu pela primeira vez na comunidade em 2002 apesar de conceito de simulação simbiótica já ter sido utilizado um pouco antes mas, era denominado por simulação *online* e este era um termo problemático visto que não tinha sido devidamente definido.

Esta é uma relação que pode ser mutuamente benéfica, por um lado a simulação que pode tirar partido da recolha de dados em tempo real sobre o sistema físico e que é garantida pelos seus

sensores, por outro lado, os sistemas físicos podem tirar partido sobre as decisões tomadas pelas simulações.

No contexto dos VANTs, Kamrani and Ayani [KA07] relacionam a simulação simbiótica com uma abordagem ao planeamento de planos dos VANTs. Outros autores avaliaram também os benefícios das suas plataformas comparando o desempenho de outras abordagens de simulação *offline*. A própria existência de um ciclo de retorno de resposta no caso de simulação simbiótica fornece aos VANTs a flexibilidade necessária através da otimização dos resultados.

Existem várias aplicações que implementam uma aproximação a este tipo de sistemas. O AgentFLY integra VANTs reais num sistema multiagentes simulado, no qual os VANTs virtuais formam uma realidade mista onde ambas as entidades coexistem. Neste projeto o planeamento e raciocínio são calculados *on-board* e a coordenação e cooperação no ambiente virtual. Outras abordagens como de Michal Jakob et al. [JPC+12] introduzem uma plataforma para desenvolvimento incremental usando dados experimentais em sistemas de múltiplos VANTs no rastreio de veículos em movimento. Outras plataformas e projetos são abordados na Secção 2.2.

Um conceito presente na simulação simbiótica é a análise de múltiplos cenários, que representam a tomada de decisões baseadas em várias alternativas simuladas. Este conceito de simulação simbiótica pode ser aplicada a uma grande variedade de domínios e com diferentes propósitos, pois consegue tirar partido das simulações para analisar sistemas físicos complexos em períodos de tempo aceitáveis.

Os cenários hipotéticos presentes na simulação simbiótica são usados para simular o comportamento do sistema físico, dependendo dos pressupostos especificados. Aydt et al. [ATCL09b] define cinco tipos distintos de sistemas de simulação simbióticos. Na Tabela 2.3 são apresentados e descritos os diferentes tipos de sistemas.

A verdade é que apesar de estes diferentes tipos de simulação simbiótica poderem ser implementados independentemente, dependendo do cenário onde se aplica, pode ainda ser considerado um último tipo de simulação simbiótica híbrida que conjuga alguns dos tipos referidos em simultâneo e a sua arquitetura encontra-se apresentada na Figura 2.5. Esta abordagem permite retirar as melhores vantagens que cada tipo de sistema oferece, por forma a poder otimizar o melhor desempenho possível.

Dadas estas possibilidades, é possível criar uma arquitetura que visa a implementação de todas as características da simulação simbiótica, permitindo criar uma arquitetura para *quadcopters* (virtuais e/ou reais) cooperativos que visam à sua implementação em aplicações do mundo real. A arquitetura, a partir do ponto de vista físico, deve apoiar os VANTs reais a partir da comunicação do estado dos sensores e das tomadas de decisão de alto nível. Contudo apesar da simulação simbiótica apresentar bastantes abordagens e vantagens, ainda se devem ter em conta os problemas dos vários tipos de sistemas e também da própria eficiência.

Um dos primeiros fatores por norma a serem postos em causa é a eficiência, uma vez que, se o próprio sistema não apresentar os resultados de forma e em tempo desejável perde a sua eficiência e o seu valor inerente ao sistema. Neste caso, cada cenário é avaliado pelos resultados da simulação e isto pode-se tornar na realidade numa tarefa muito complexa com muitos problemas. Também

Tabela 2.3: Tipos de Sistemas de Simulação Simbiótica

Tipo	Funcionalidade	Descrição
SCSS	Controlo de sistemas físicos	Este tipo de sistema controla diretamente o sistema físico através dos controladores, ou seja, o próprio controlador pode ser ativado, delegando ao sistema físico para executar determinadas ações.
SSDSS	Suporte de tomada de decisões externo	Neste tipo de sistemas as decisões são tomadas por um "decisor" externo que pode implementar as decisões no sistema físico. O próprio sistema beneficia dos dados em tempo real, que pode ser injetado nas simulações em execução por forma a aumentar a eficácia dos resultados finais.
SPSS	Previsão de um sistema físico	Este sistema tenta explorar possíveis previsões do estado do sistema refletindo sobre diferentes hipóteses tendo o ambiente em conta, por exemplo um sistema anti colisões num conjunto de VANTs é uma possível aplicação deste tipo de sistema.
SVMSS	Validação de um sistema de simulação	Neste sistema, os comportamentos obtidos através da simulação são comparados com o comportamento do sistema físico real. Então a hipótese que mais se aproximar é considerada como o modelo de referência, em outras palavras, este tipo de sistema pretende validar o modelo virtual e as respetivas estratégias de controlo.
SSADS	Detetor de anomalias em sistemas físicos ou modelo de simulação	Neste sistema é feita uma análise contínua à procura de discrepâncias entre os sistemas reais e virtuais que são consideradas anomalias. Este tipo de sistemas podem ser usados para detetar anomalias quer nos sistemas físicos como nos modelos virtuais. Este processo passa pela comparação do comportamento simulado com o comportamento atual. Pode também ser utilizado para detetar comportamentos indesejados no sistema.

costumam surgir problemas relacionados com os modelos dos sistemas reais, uma vez que, é necessário que se implementem os modelos com o maior nível de precisão possível, para que se consigam tomar as melhores decisões em relação ao sistema real. A aplicação de pequenos ruídos nos cálculos pode levar a repercussões e por sua vez a grandes desvios dos valores esperados. Um método que tentou anular estes ruídos foi a utilização da simulação do sistema físico em tempo-real tentando anular assim esses desvios, estando este método descrito em Low et al. [LLL$^+$05].

2.2 Estado da Arte

Nesta secção pretende-se descrever trabalhos e investigações relacionados com o tema da dissertação que tenham objetivos similares ou que se encontrem dentro da mesma área de interesse.

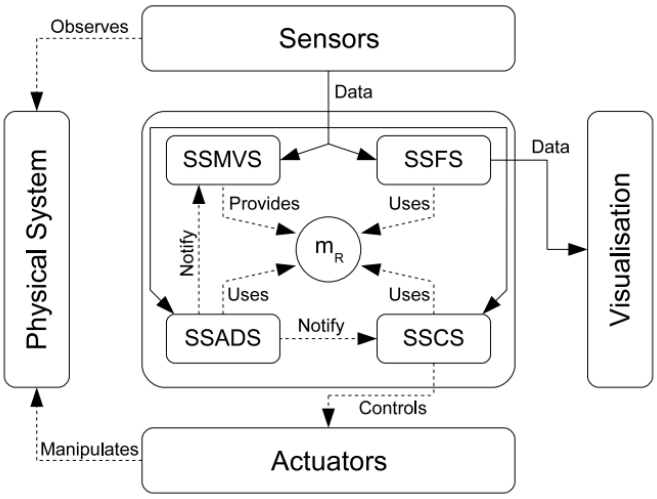


Figura 2.5: Simulação Simbiótica Hibrida [ATCL08]

Na Tabela 2.4 são apresentados todos os projetos que foram analisados ou identificados durante leitura dos vários artigos.

De um modo geral, foram analisados alguns projetos de *quadcopters* autónomos dos quais se conseguiu perceber o seu modo de funcionamento e construção. Ainda nos trabalhos estudados estão presentes sistemas de simulação que implementam vários tipos de arquitetura, como a simulação simbiótica e co-simulação. Estes vão ser apresentados nas seções seguintes e serão analisados do ponto de vista das contribuições e relações com a presente dissertação, ou seja, o que cada plataforma possui de bom para o desenvolvimento desta nova plataforma e os problemas que ainda apresentam.

2.2.1 *Quadcopters* Autónomos

Dado o tema da dissertação residir sobre o desenvolvimento de uma plataforma de desenho, simulação e desenvolvimento de SMA de *quadcopters*, os critérios utilizados para selecionar os trabalhos relacionados, foram a capacidade de perceção e a autonomia nos veículos autónomos uma vez que estes devem ser capazes de interagir e de se mover num ambiente independentemente dos obstáculos que encontrem.

Estes veículos autónomos no âmbito da dissertação são denominados *quadcopters* e pertencem ao grupo dos VANTs. A Tabela 2.5 apresenta uma lista com alguns projetos que têm vindo a ser desenvolvidos ao longo destes últimos anos com os respetivos locais de desenvolvimento e a técnica que implementam e que lhes garantem um certo nível de autonomia.

Começando pelo STARMAC [HRW+04] da Universidade de Standford, este projeto implementa um controlador que se baseia numa técnica de aprendizagem por reforço, no qual o VANT (neste caso o agente) executa as suas ações num ambiente de forma a tentar maximizar alguma noção de recompensa cumulativa. O uso desta técnica na plataforma pode ser uma mais-valia para

Tabela 2.4: Projetos Relacionados

Quadcopters Autónomos	**Simuladores de VANTs**	**Simuladores com Mixed Reality**	**Co-simuladores**
STARMAC	Flight Simulation	USARSim	JOCosim
OS4	SPEEDES	Webots	OFFIS SimLink
Helio-copter	Virtual Reality Toolbox	Microsoft Robotics Studio simulator	TTMC
HMX-4	Piccolo	Gazebo	IntelWheels (LIAC-C/FEUP)
Quad-Rotor UAV	Microsoft Flight Simulator		
Quad-Rotor Flying Robot	X-Plane		
	FlightGear		

também tentar maximizar de alguma forma os possíveis resultados esperados a partir da múltipla repetição dos mesmos cenários.

Uma outra abordagem é utilizada no projeto OS4 [BNS04], neste projeto é utilizado um controlador de *backstepping* combinado com um controlador baseado em regras que permitem novas abordagens na atitude da estabilização do VANT. Segundo os autores, esta abordagem permite retirar uma melhorar eficácia do que na utilização da técnica de *backstepping* de forma isolada. Apesar de esta técnica indicar que apresenta tempos de resposta lentos, a mesma poderia ser utilizada para uma contribuição da nova plataforma. A ideia passaria por utilizar a mesma técnica para a criação de um controlador de posição do VANT, uma vez que nesta abordagem não seriam necessárias respostas imediatas mas sim uma resposta à deslocação do VANT para a posição desejada.

O projeto do Helio-copter [TLFA09] tem uma abordagem um pouco limitada, devido a estar relacionada com técnicas que implementam visão. Esta seria uma abordagem a utilizar numa fase mais avançada da plataforma, em que se instalariam novos sensores como câmaras para que os agentes pudessem analisar as imagens e executar as respetivas tomadas de decisão. À semelhança deste projeto, o HMX-4 [ZLWZ11] também utiliza os dados das câmaras no seu controlador de linearização, em ambientes controlados assim como o Microraptor [RYAS09]. Este último apenas difere na navegação pois utiliza caminhos pré-definidos.

Finalmente, temos a utilização do controlador proporcional integral (PI) e do controlador proporcional integral derivativo (PID) no Quad-Rotor Flying Robot [WA06]. Este último é uma boa abordagem para uma primeira implementação na plataforma de simulação de *quadcopters*. O controlador permite dar uma resposta mais rápida, estável e com mais precisão de acordo com os dados de entrada que são fornecidos. Na maior parte das vezes estes mesmos controladores são utilizados nos controladores eletrônicos em ciclo fechado.

Tabela 2.5: Quadcopters Autónomos [AMM06]

Nome	Universidade	Técnica Utilizada
STARMAC [HRW+04]	Standford University	Reinforcement Lrn.
OS4 [BNS04]	EPFL	Backstepping
[Kiv06]	Pennsylvania State University	PI
Helio-copter [TLFA09]	Brigham Young University	Visual Feedback
HMX-4 [ZLWZ11]	Pennsylvania State university	Feedback Lin.
Quad-Rotor UAV [CH03]	University of British, Columbia, Chen and Huzmezan	MBPC and H∞
Quad-Rotor Flying Robot [WA06]	Universiti Teknologi Malaysia	PID
Microraptor [RYAS09]	Oakland University	Autonomous predefined path navigation and image acquisition.

2.2.2 Simuladores de VANTs

Nesta secção vão ser apresentados alguns simuladores de VANTs não especificando em concreto sobre os *quadcopters*, ou seja, não necessita de ser exatamente um *quadcopter* visto todos os VANTs partilharem os mesmos conceitos básicos. Na Tabela 2.6 são apresentados alguns simuladores de VANTs cada um com uma breve descrição da plataforma e de seguida são analisados mediante o problema que se pretende resolver.

A grande parte dos trabalhos mais conhecidos são os simuladores de voo, no qual integram interfaces gráficas bastante realistas. Os estudos indicam que se deve ter alguma atenção em relação às dinâmicas de voo uma vez que a maioria das plataformas implementam a mesma dinâmica de voo para todos os modelos, ou seja, não implementam muita precisão relacionada com as diferenças que cada veículo possa ter. O foco desta análise passa pelo estudo do funcionamento da plataforma, as suas funcionalidades e a capacidade de se poder integrar com outros sistemas, descartando a linguagem de programação utilizada, mas tendo em atenção a interface de ligação disponibilizada.

Começando pela plataforma do Flight Simulation [JF01] esta apresenta uma estrutura para a modelação da plataforma bastante simples, dividindo a plataforma no modelo do veículo, modelo do controlador e o modelo de localização no ambiente. Esta estrutura permitiu verificar que se deve então dividir o modelo do veículo e o modelo do ambiente de formas distintas assim como a implementação do controlador.

A plataforma SPEEDES [CL04] é um simulador que implementa uma abordagem para execução de processamento em paralelo de forma sincronizada. Esta seria também uma boa abordagem para a plataforma de simulação no caso de se implementar a agentes distribuídos e em grandes quantidades de instâncias. Seria uma mais-valia para dar resposta à necessidade de resultados bastante otimizados e em pouco tempo. Virtual Reality Toolbox [CNF09] é um projeto orientado mais para a implementação de estratégias de coordenação do que para a apresentação gráfica, não

deixa de ser uma plataforma de simulação e também uma boa abordagem para a implementação destas estratégias nos próprios agentes.

Quanto ao Microsoft Flight Simulator [GSRO08] também foi analisado por ser bastante conhecido, apesar de ser considerado por muitos como um jogo em vez de um simulador. Esta plataforma integra a modelação de falhas nos equipamentos dos VANTs mas também de uma forma genérica e sem precisão. Também apresenta uma API que permite a integração com outros sistemas, facto pelo qual chamou a atenção que se poderia utilizar para a representação 3D invés de se estar a desenvolver uma de raiz. No entanto este só permite o controlo de um VANT de cada vez e a nível de sensores estaria muito limitado.

À semelhança do Flight Simulator, o X-Plane [GSRO08] e FlightGear [GSRO08] também permitem a integração com outros projetos em desenvolvimento. Ambos são utilizados como plataformas de investigação em projetos pessoais ou até mesmo em ambientes académicos, no entanto também são limitados em relação ao ambiente e à implementação de sensores.

Tabela 2.6: Simuladores de VANTs

Nome	**Descrição**
Flight Simulation [JF01]	Esta é uma ferramenta de simulação que foi desenvolvida em C e inclui três modelos principais na sua estrutura, modelo do veículo, modelo de interface servo e modelo GPS. Este simulador reduz o risco e a quantidade de voo de testes para a certificação dos sistemas de voo mais complexos.
SPEEDES [CL04]	O simulador *Synchronous Parallel Environment for Emulation and Discrete-Event Simulation* (SPEEDES) permite executar processamento paralelo otimista em computadores de alta performance, redes, workstations ou em combinações de computadores em rede.
Virtual Reality Toolbox [CNF09]	É uma interface que fornece cenários típicos associados a manobras de reabastecimento. Permite, através da simulação, guiar o VANT até à posição e orientação associada aos objetos no ambiente virtual.
Simulink [GSGA09]	É uma plataforma de simulação que permite recolher dados sobre os voos efetuados. Assim permite a verificação dos mesmos a fim de se obter a informação necessária para os ajustes dos parâmetros do controlador.
Adriano Mancini et al. [MCI$^+$09]	Plataforma baseada em modularidade e estruturada em camadas específicas. Permite a fácil permuta do ambiente simulado para o real, reduzindo assim os tempos de testes e *debug*.
Microsoft Flight Simulator [GSRO08]	É um dos simuladores de voo mais conhecido do mercado. É baseado em modelos de voo constituídos pelos veículos e pelos ambientes genéricos. Integra modelação de falhas nos equipamentos de uma forma genérica e sem precisão.
X-Plane [GSRO08]	Este simulador usa nos seus modelos uma forma geométrica no qual divide um veículo em subdivisões mais pequenas no qual permite determinar as forças em cada posição. É um simulador profissional e está aprovado pela Administração Federal de Aviação (AFA) dos Estados Unidos para certificação de treino.
FlightGear [GSRO08]	Este é um simulador de código-fonte aberto, multiplataforma e permite a cooperação do simulador de voo em projetos em desenvolvimento por forma a poder ser utilizado como plataforma de investigação.
Piccolo [GSRO08]	Piccolo é um sistema de piloto-automático para veículos de pequena dimensão para permitir voos autónomos e integra também um ambiente de simulação.

2.2.3 Simuladores com *Mixed Reality*

Nesta subsecção vão ser listados os sistemas reais/virtuais, ou seja, os sistemas onde exista uma relação simbiótica entre o agente real e virtual. O conceito da combinação do mundo real com o virtual surge em diversos domínios de aplicação [ROB07], mas nestes trabalhos a característica predominante é que deve existir uma troca de informação entre o *quadcopter* real e o *quadcopter* virtual que o representa no simulador. Isto é, devem ser capazes de trocar informações

em tempo real para recolha de dados dos sensores, quer para envio de tomada de decisões para os controladores ou para as duas situações em simultâneo. A Tabela 2.7 apresenta alguns simuladores que integram *Mixed Reality* com uma breve descrição da plataforma.

Outros modelos podem ser encontrados, como por exemplo o *Polyagents* que é um novo modelo que representa cada entidade como um conjunto de agentes. Este utiliza o sistema para dar suporte nas tomadas de decisão dos modelos virtuais, cada entidade possui um conjunto de agentes iguais denominados "fantasmas" para assim poderem explorar as várias alternativas, utiliza uma técnica de simulação de múltiplas trajetórias onde se calculam simultaneamente várias alternativas de uma só vez. Esta abordagem permite avaliar os vários resultados possíveis em cada momento selecionando os mais vantajosos e propagando-os aos agentes [DPBW$^+$08].

O USARSim [CLW$^+$07] é uma plataforma indicada para a simulação de robôs, baseada no motor gráfico do Unreal e permite a instanciação dos modelos dos robôs com alguns sensores. Ainda permite o acesso aos robôs do mundo real através de um protocolo de comunicação proprietário. Esta plataforma reúne algumas das funcionalidades necessárias para a nova plataforma de simulação de *quadcopters* autónomos. O Webots [Mic04], também é muito semelhante distinguindo-se pela configuração de robôs muito mais complexos respeitando toda a parte mecânica de movimentos e forças. Esta situação também é pretendida no que diz respeito à aplicação dos forças geradas pelos componentes dos *quadcopters*.

Uma outra ferramenta mais profissional é o Microsoft Robotics Studio Simulator [Jac07]. Esta plataforma implementa vários tipos de arquiteturas e no geral é uma ferramenta que apresenta algumas dificuldades na integração com outros projetos apesar de estarem a tentar introduzir esta ferramenta em ambientes académicos. Por fim, apresenta-se o Gazebo [KH04], uma plataforma de código fonte livre e bastante completa, que permite a instanciação de múltiplos robôs e a criação de ambientes complexos.

Todas estas plataformas apresentadas nesta secção reúnem funcionalidades necessárias para a implementação da plataforma, no entanto nenhuma delas tem todas as funcionalidades necessárias para fazer a sua integração. Por exemplo, nem todas permitem a importação de modelos, outro problema é o facto de só o USARSim e o Gazebo darem suporte na integração de agentes externamente. Porém todos apresentam a implementação de motores de física, uns mais robustos que outros. Uma outra vantagem destas plataformas em relação às plataformas de simulação de VANTs é que as anteriores já integram alguns sensores como sonares e infravermelhos [PMRR11].

Tabela 2.7: Simuladores com *Mixed Reality*

Nome	**Descrição**
USARSim [CLW+07]	USARSim é um simulador de código fonte aberto, mais indicado para a simulação de robôs que tanto pode ser utilizado para investigação como para fins educativos. Distingue-se dos restantes simuladores por ser utilizado em competições de robôs virtuais.
Webots [Mic04]	Simulador de dispositivos robóticos complexos incluindo toda a parte mecânica que requer a simulação das leis da física. Utiliza Open Dynamics Engine (ODE) na simulação da física.
Microsoft Robotics Studio simulator [Jac07]	É uma plataforma que pretende vir a ser um padrão para o controlo de robôs. Implementa arquiteturas para concorrência, distribuição, abstração, simulação entre muitas outras. Também tem como objetivo ser algo de uso por parte das comunidades académicas.
Gazebo [KH04]	Este simulador é desenhado para criar ambientes 3D para múltiplos robôs, capazes de implementar mundos virtuais complexos. Tem código-fonte aberto, permite a simulação de ambientes remotos e ainda a capacidade de fazer engenharia reversa de sistemas *black-box*, no qual se desconhece o seu funcionamento interno.

2.2.4 Co-simuladores

Os trabalhos de co-simulação neste documento são considerados, como tendo a característica principal de conseguirem simular o seu *hardware*, mais propriamente os sensores, ou seja, é simulado todo o lado físico do equipamento real. A simples capacidade de se conseguir simular o funcionamento dos sensores agrega um valor importante nas próprias simulações uma vez que não estejam disponíveis os equipamentos na realidade.

À semelhança de outros projetos, por exemplo, na co-simulação com veículos elétricos onde se utiliza uma arquitetura desenvolvida que integra dois simuladores em conjunto (Simulink e SUMO). Conseguiram retirar melhores análises do fluxo de tráfico urbano e efeitos dinâmicos na eficácia dos autocarros elétricos, onde se co-simulou o veículo no nível mais alto de abstração e o seu motor físico num nível mais baixo [MKSP13].

Na Tabela 2.8 são apresentados alguns projetos relacionados com a co-simulação. Na sua maioria são utilizados modelos matemáticos do MATLAB para a geração dos dados de saída dos sensores. Esta funcionalidade é bastante importante para a integração da plataforma de simulação, à semelhança do projeto IntelWheels, em que também são gerados dados virtuais usados para reproduzir o comportamento da cadeira de rodas real no teste dos controladores. Este mesmo conceito pode ser utilizado para a integração dos *quadcopters*, permitindo simular o funcionamento de sensores como os infravermelhos, sonares e até mesmo os motores utilizando controladores PID que controlam as rotações dos motores.

Tabela 2.8: Co-simuladores

Nome	Descrição
JOCosim [AMM06]	É um co-simulador baseado em linguagem de programação C e a versão mais recente já é implementada em Java. O Java consegue controlar as simulações independentemente de utilizar as simulações fornecidas pelo OPNET na versão anterior. Desta forma, consegue integrar outras ferramentas como por exemplo o MATLAB. Esta plataforma no presente artigo foi utilizada para a análise de grupos de VANTs.
OFFIS SimLink	Plataforma de código fonte aberto que integra co-simulação através do Matlab e Simulink. O sistema e o ambiente são implementados como modelos executáveis, a escolha da estratégia adotada recai sobre os resultados do elevado número de simulações efetuadas.
TTMC [JCWW10]	É uma plataforma de co-simulação para 3D Network-on-Chip, que permite observar o fluxo e a temperatura nos circuitos integrados.
IntelWheels (LIACC/-FEUP) [BPMR08]	É uma cadeira de rodas com um elevado nível de cognição desenvolvido utilizando SMA e Simulação. No modo de simulação o sistema funciona apenas com os dados virtuais e são usados para dois fins: gerar o mesmo comportamento de uma cadeira de rodas real e para testar as rotinas de controlo.

2.3 Conclusão

Este capítulo, estando relacionado com o estudo das áreas necessárias para o desenvolvimento do projeto, revelou-se bastante extenso e complexo, devido em grande parte à quantidade de informação e de literatura disponível. Isso dificultou em certas ocasiões uma fácil interpretação da teoria que os autores tentam passar, uma vez que alguns dos termos eram bastantes similares entre si em relação à Simulação.

O uso dos SMA em conjunto com a simulação permite trazer mais vantagens, quer para treinos, quer para otimizações de algoritmos ou de execução de tarefas. Assim como o uso da simulação simbiótica que consegue retirar vantagens do que cada um dos mundos, virtual ou real, oferece de melhor. Analisados os SMA e a Simulação Simbiótica, pode-se desde logo correlacionar estes dois tipos de sistemas. Dado os agentes serem entidades autónomas que conseguem cooperar de modo a atingir determinados objetivos ou metas comuns, este tipo de sistema consegue suportar todos os aspetos relacionados com a sinergia entre a simulação e os domínios reais.

Várias são as investigações para a implementação de arquiteturas de simulação de veículos autónomos [PR12] como também são várias as possibilidades das aplicações que podem ser dadas aos agentes na plataforma. Por exemplo, podem ser utilizados para se deslocarem a armazéns e recolherem determinados objetos para procederem a entregas, isto em armazéns de fármacos [JKRM12], correios, de livros como a Amazon, etc.

Apesar de já existirem algumas plataformas de simulação simbiótica, será de esperar que comecem a surgir cada vez mais ao mesmo tempo que se tentam desenvolver novos e sofisticados

algoritmos e técnicas para tentar resolver todos os problemas que surgem na realidade, no qual estes sistemas possam contribuir com os seus resultados e otimizações.

Capítulo 3

Desenvolvimento

Neste capítulo vai ser apresentado o desenvolvimento da solução final, assim como cada passo da sua implementação, desde as tecnologias utilizadas à forma como foram implementados todos os objetivos. Destacando também os requisitos definidos, os pressupostos e a arquitetura geral para uma melhor orientação de todo o desenvolvimento da plataforma.

3.1 Requisitos

Os requisitos funcionais da plataforma devem ser completos, no sentido de representar todas as características das funcionalidades pretendidas. Devem ser também consistentes para não existirem incoerências entre eles e com os pressupostos definidos. Na especificação dos requisitos, vão ser descritos de acordo com o foco do projeto e numa forma simplificada por forma a reduzir as ambiguidades que possam surgir através do uso de notações gráficas [Jac99].

Requisitos Funcionais

Com o objetivo de assegurar o funcionamento essencial da plataforma de simulação, é necessário que o sistema cumpra os seguintes requisitos funcionais apresentados na Tabela 3.1. Estes encontram-se divididos em macro requisitos e micro requisitos, sendo descritos logo de seguida:

- **Simulação de VANTs**: Deve-se implementar um modelo gráfico e analítico do *quadcopter* juntamente com o modelo virtual do cenário, onde estes devem interagir de forma autónoma. O modelo analítico deve representar o modelo do *quadcopter* real através da configuração de parâmetros de acordo com os dados dos fabricantes dos componentes.

  – **Múltiplos VANTs**: Deve permitir a instanciação de múltiplos *quadcopters* virtuais e/ou reais, todos estes com a visão do mesmo mundo, sendo este um requisito crucial para se garantir um sistema multiagente.

Tabela 3.1: Requisitos

Plataforma de Simulação de VANTs c/Ligação Simbiótica			
Macro Requisitos	• Simulação de VANTs	• VANTs Autónomos	• Simulação Simbiótica
Micro Requisitos	– Múltiplos VANTs	– 2 Camadas	– Comunicação
	– Dinâmicas	– Linguagem de Alto Nível	– Sensores
	– Validação	– Deliberativos	
	– Estabilidade		

– **Dinâmicas**: Deve implementar o funcionamento do *quadcopter* no que diz respeito às leis da física que estes apresentam, num determinado ambiente virtual.

– **Validação**: Deve-se implementar os métodos necessários para validar os modelos teóricos quer através dos resultados ou por observação dos comportamentos executados.

– **Estabilidade**: A estabilidade é outro requisito que se deve implementar para se ter um melhor controlo do VANT na execução das missões, ao nível do controlo dos motores.

- **VANTs Autónomos**: É necessário implementar um motor de agentes com uma arquitetura, por exemplo, BDI por forma a se poder tirar partido da cognição dos agentes para a tomada de decisões e execução dos planos permitindo uma abordagem diferente dos VANTs atuais que é reativa.

  – **2 Camadas**: Deve ser aplicado uma arquitetura a duas camadas, dividindo a carga computacional com o trabalho da implementação entre as duas soluções distintas. No caso específico seria utilizado a placa controladora CC3D OpenPilot para controlar as funções básicas do *quadcopter* no nível mais baixo, e uma plataforma genérica para definir o centro de decisões e o controlo de alto nível do agente VANT.

  – **Linguagem de Alto Nível**: Outro requisito importante é a necessidade de se implementar os planos dos agentes numa linguagem abstrata de alto-nível por forma a se conseguir dar o máximo de utilidade à plataforma sem a necessidade de se modificar o seu código. Este requisito deve passar pela integração de um interpretador de planos de forma a executar os planos que lhes forem desenhados e atribuídos.

  – **Deliberativos**: Permitir que os agentes sejam deliberativos, uma vez que no ambiente simulado tem conhecimento de uma parte do ambiente e assim se poder tomar as melhores decisões a seguir (sem nunca sobrepor a camada reativa).

- **Simulação Simbiótica**: Deve integrar uma ligação simbiótica que deve existir entre um *quadcopter* virtual e um *quadcopter* real. De acordo com os paradigmas de simulação simbiótica referidos no Capítulo 2 deve existir uma ligação simbiótica entre um *quadcopter* real e um virtual por forma a conseguir-se retirar o que cada um dos mundos oferece de melhor.

– **Comunicação**: Para este requisito deve ser integrado o meio de comunicação necessário para que se consiga trocar informação entre o sistema virtual e o real, no qual se pode utilizar os módulos de rádio XBee.

– **Sensores**: Simular os sensores infravermelhos e os sonares é outro requisito necessário para a co-simulação emulando o funcionamento dos sensores reais e garantindo assim as perceções relacionadas com a localização e deteção de obstáculos.

Este conjunto de requisitos funcionais especificados é significativo para a arquitetura geral do sistema apresentada na secção 3.3. Arquitetura esta que não é mais do que uma solução para o problema de investigação definido na secção 1.2.

Requisitos Não Funcionais

Os requisitos não funcionais da plataforma estão relacionados ao uso da aplicação em termos de desempenho, usabilidade, fiabilidade e escalabilidade. Caracterizam o sistema de um ponto de vista mais genérico, sendo fatores determinantes na determinação da qualidade da plataforma. Assim, de seguida são apresentados esses requisitos que o sistema deverá cumprir.

Ao nível da utilização as características da interface do utilizador, deve-se permitir uma fácil configuração dos agentes, ou seja, deve-se ter uma interface simples, amigável e intuitiva. Isso através da clara apresentação de todos os campos e botões.

No que diz respeito à fiabilidade, a plataforma deve garantir a precisão dos cálculos efetuados com uma taxa de falhas aceitável. Quanto ao desempenho, esta deve garantir respostas em tempo útil das mais variadas tarefas, assim como garantir a velocidade de processamento com os vários agentes. Por fim deve dar suporte na escalabilidade da plataforma e depuração de dados de simulação.

3.2 Pressupostos

Alguns pressupostos foram definidos com base nos requisitos do projeto, visto que embora a implementação possa permitir mais funcionalidades, não vão ser consideradas determinadas situações pelo que se destacam de seguida as mais relevantes.

Em primeiro lugar a plataforma deve ser capaz de simular o modelo do *quadcopter*, isto quer dizer que deve conseguir realizar as manobras com o maior grau de precisão possível de acordo com o real. Para tal, o ambiente onde estes modelos vão ser integrados deve ser estático, deste modo consegue-se uma análise mais rigorosa dos dados recolhidos e ficam menos suscetíveis a erros durante todo o desenvolvimento.

No Apêndice A descreve-se a instalação do ambiente de desenvolvimento, desde a análise das tecnologias disponíveis à seleção da linguagem de programação e ferramentas de desenvolvimento utilizadas.

Os cenários de testes escolhidos serão definidos no Capítulo 4, e encontram-se relacionados com ambientes controlados como salas, auditórios, locais onde se consigam anular efeitos do

vento e objetos em movimento. Conseguindo desta forma, retirar resultados do sistema físico e comparar ambos os dados de forma mais precisa possível.

Deve-se considerar uma abordagem simples com capacidade de coordenação dos movimentos básicos, tais como frente, esquerda, direita, e outros. Numa fase inicial serão utilizados através do controlo humano para a realização do teste dos movimentos no modelo do simulador, para posteriormente se utilizar os agentes na deslocação autónoma. Verificou-se a necessidade de considerar também emuladores válidos para os sensores e controladores dos motores do *quadcopter* para que os agentes sejam autónomos.

Este simulador deve ser escalável, o que quer dizer que deve suportar múltiplas instâncias de *quadcopters* virtuais no ambiente em simultâneo. Outro pressuposto definido foi a utilização do controlo remoto do *quadcopter* através da plataforma de simulação, permitindo assim numa fase inicial evitar a danificação do equipamento real.

Todos os *quadcopters* no final do projeto devem ser autónomos implementando planos simples como deslocar do local A para um local B onde se consigam evitar colisões e obstáculos. Neste caso poderão ser implementados Sistemas Anti Colisão de Tráfego (SACT) numa forma mais simplificada, no qual através da troca constante de informações é possível construir um mapa tridimensional dos VANTs com informações tais como: localização, rumo, altitude e velocidade. Utilizando essas informações, o sistema pode calcular as posições futuras de todos os VANTs e assim determinar-se os potenciais riscos de colisão [KD07].

Deste modo, eles devem ter a perceção do ambiente que os rodeia com a indicação do seu estado interno e posição, para conseguirem concretizar as suas tarefas de uma forma eficaz e controlada. Por fim, deve existir comunicação entre duas instâncias de *quadcopters* ou mais, onde por sua vez consigam interagir entre *quadcopters* virtuais e também entre o mundo virtual e o real.

Por último, resta referir que os cálculos analíticos das forças, potências, RPMs dos motores e dos restantes cálculos dos componentes são apenas teóricos e está implícita uma pequena perda de precisão. Esta perda de precisão deve-se à dificuldade de reunir e quantificar todas as variáveis que podem influenciar os resultados como, por exemplo, a temperatura, atritos, e outros fatores dinâmicos desconhecidos nas respetivas fórmulas.

3.3 Arquitetura Geral

A arquitetura apresentada na Figura 3.1 representa a globalidade do projeto, demonstrando a interação entre o sistema simulado e o sistema físico. Foi seguido um paradigma de simulação simbiótica, no qual um *quadcopter* virtual (simulado) e um real (físico) se encontram fortemente ligados. Neste paradigma podem-se facilmente retirar benefícios mútuos da sinergia presente. Por exemplo, o simulador pode beneficiar dos dados dos sensores enquanto o sistema físico beneficia das conclusões obtidas dos resultados das simulações, ou ainda para se utilizar a ligação simbiótica para efetuar uma simulação com informação obtida em tempo real e desta forma auto ajustar às condições simuladas.

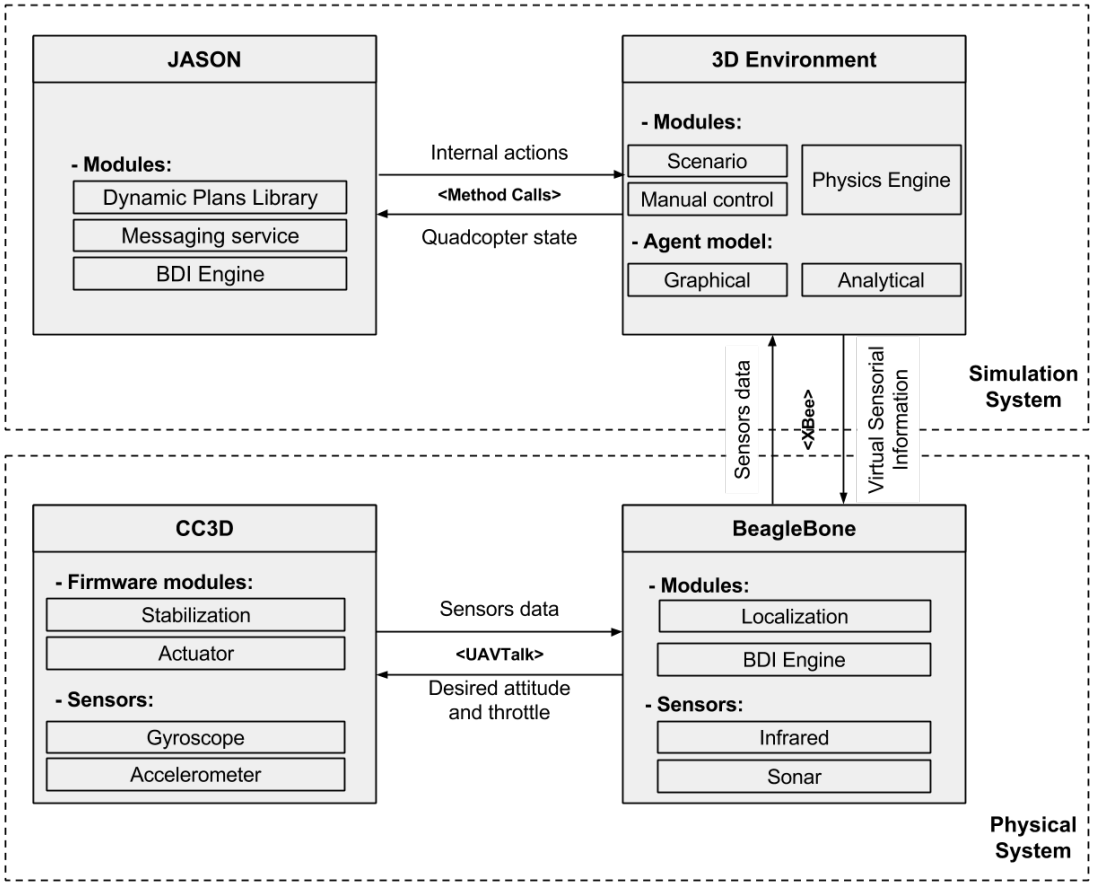


Figura 3.1: Arquitetura Geral

Com esta arquitetura pretende-se explorar então vários tipos de funcionamento do sistema com o objetivo de controlar um *quadcopter* real, dar suporte na tomada de decisões, treinar o *quadcopter* em diferentes ambientes, validar o modelo virtual e estratégias, e ainda detetar anomalias tanto no sistema simulado como no físico. Dadas todas estas possibilidades, esta arquitetura permite a integração de todas as funcionalidades da simulação simbiótica acima referidas, na cooperação entre *quadcopters* com o objetivo da sua utilização em aplicações do mundo real.

Ainda na arquitetura, da perspetiva do sistema físico, este deve dar suporte para a comunicação entre estes dois sistemas, podendo assim enviar dados do estado dos sensores de forma fiável e receber comandos das tomadas de decisão de alto nível.

O ambiente virtual deve permitir a representação realista dos modelos, uma vez que as simulações geradas podem ser usadas para treino de voo, teste de algoritmos de controlo ou ainda na coordenação, cooperação e colaboração dos agentes. O **Sistema Simulado** é uma plataforma que se encontra dividido em duas componentes: a primeira representa a integração do processo de tomadas de decisões dos agentes em alto nível e a segunda componente é a responsável pelo controlo de baixo nível e representação gráfica.

Um dos principais objetivos é facilitar o modo como se implementam os planos para os

VANTs, dai ter sido utilizado o Jason. Uma extensão de uma conhecida linguagem de programação orientada aos agentes conhecida como *AgentSpeak*, para se poder desenvolver os comportamentos em alto nível sem a necessidade de implementar código novo [HBA11, RHB07, BH06]. Cada agente *AgentSpeak* é definido por um conjunto de crenças, objetivos e por planos que adotam o modelo (BDI) crenças, desejos e intenções. Jason é um interpretador que permite a adaptação dos planos em tempo de execução para que os próprios agentes consigam melhorar as suas reações e flexibilidade através dos vários cenários de simulação que sejam apresentados.

Uma outra funcionalidade do *Jason*, não menos importante, é a capacidade de comunicação interna entre os agentes. Isto permite aos agentes no mesmo ambiente virtual e também no ambiente real através do paradigma apresentado coordenar, cooperar e colaborar. Esta comunicação é feita através do *speech-act*, um protocolo de comunicação entre agentes incluído no *AgentSpeak*.

Os agentes no *Jason* reagem no ambiente virtual através de comandos de alto nível, onde por sua vez o ambiente as vai executar em comandos de baixo nível. A solução passa por criar umas ações internas que permitem a ligação do agente ao ambiente. Estas ações internas devem ser sempre desenvolvidas da perspetiva do agente, uma vez que estas serão executadas por si através do motor interno do *Jason* onde as semânticas dos planos são interpretadas. Esta implementação expande as funcionalidades das ações internas através de uma interface entre o *Jason* e o ambiente virtual implementado em *jME3*. Desta forma os agentes sabem exatamente que estão a enviar as ações para os respetivos comandos de baixo nível sem engano.

O ambiente 3D usa o jMonkeyEngine *jME*, um conjunto de ferramentas de desenvolvimento de software baseado em java e encontra-se dividido em três módulos essenciais: o cenário, o motor de física, e o controlo manual. O primeiro módulo vai suportar a representação 3D do ambiente simulado com o objetivo de proporcionar as perceções para os agentes virtuais e reais. O *jME3* também já inclui um motor de física bastante robusto, que consiste numa extensão do *jBullet* para se testar o controlo e a estabilidade dos VANTs. O módulo de controlo via teclado ou controlo por rádio também se encontra incluído exclusivamente para fins de testes e intervenções humanas que sejam necessárias.

Em relação ao modelo do agente este encontra-se dividido no modelo gráfico, onde se pode definir um modelo genérico de um *quadcopter* ou importar um modelo 3D existente do *quadcopter* real. Também se encontra dividido no modelo analítico, onde todos os parâmetros ou configurações disponíveis devem corresponder aos dados técnicos recolhidos das fichas técnicas dos fabricantes de todos os componentes presentes no *quadcopter*.

Existem duas interfaces no sistema simulado: uma interna entre o *Jason* e o *jME3*, e uma externa entre o simulador e o sistema físico através dos módulos de rádio XBee. A primeira consiste nas já referidas ações internas e, na direção oposta, o ambiente transmite para o *Jason* o estado de todos os agentes que resultam nas próprias perceções. A última é implementada fisicamente através dos módulos XBee que estabelecem uma ligação rádio para enviar dados sensoriais virtuais e também para receber informação dos sensores ou comandos do *quadcopter* real.

Apesar de o **Sistema Físico** não ser o foco desta dissertação, faz-se uma breve descrição uma vez que este sistema também é apresentado na Figura 3.1. É utilizada a mesma filosofia aplicada no

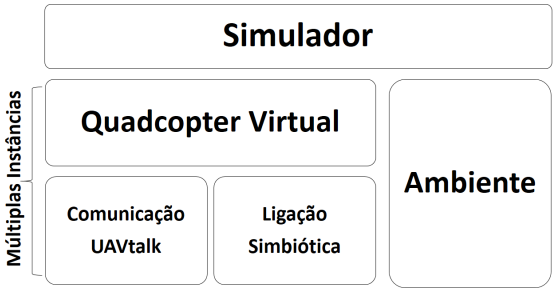


Figura 3.2: Arquitetura Conceptual da Plataforma de Simulação

Sistema Simulado que separa as tarefas de alto e baixo nível do agente, só que neste caso o agente é incorporado no *quadcopter* real. Tirando partido das plataformas eletrônicas existentes este é baseado na OpenPilot CC3D para estabilização e controlo dos motores enquanto os processos de tomadas de decisões do agente é processado na plataforma BeagleBone.

3.4 Meta-arquitetura dos Agentes VANT

A modelação computacional é um campo multidisciplinar de conhecimento que trata da aplicação de modelos matemáticos e técnicas de análise de computador, compreensão e estudo da fenomenologia de problemas complexos em áreas tão amplas como a engenharia, ciências e muitas outras. Na Figura 3.2 apresenta-se a arquitetura conceptual da plataforma de simulação numa vista mais simplificada. O simulador encontra-se dividido em dois módulos distintos sendo o primeiro módulo referente à implementação dos *quadcopters* que vão ser abordados nesta secção. O segundo módulo refere-se ao ambiente onde todos os *quadcopters* vão se encontrar integrados, e que será apresentado posteriormente na secção 3.5.

A meta-arquitetura do agente VANT consiste no desenvolvimento de um modelo analítico, capaz de representar um VANT com todas as especificações e componentes que estarão presentes no sistema físico. Para além do modelo analítico, é acrescentado um modelo gráfico para a representação do *quadcopter* no ambiente 3D. Cada um destes dois modelos vai integrar todos os objetivos que foram previamente definidos, que serão abordados mais em pormenor nos tópicos seguintes. Vão ser analisados os problemas que possam surgir e as formas de os resolver, focando sempre o desenvolvimento de uma plataforma capaz de simular um *quadcopter* real com grande fiabilidade.

3.4.1 Modelo Analítico

O modelo analítico vai ser composto principalmente por quatro componentes básicos, como se apresenta na Figura 3.3. O primeiro componente é o modelo elétrico do VANT, que deve ser composto por todas as características presentes em um VANT real, a partir das características técnicas apresentadas nas folhas de dados técnicos sobre motores, quadros, hélices, *etc*.

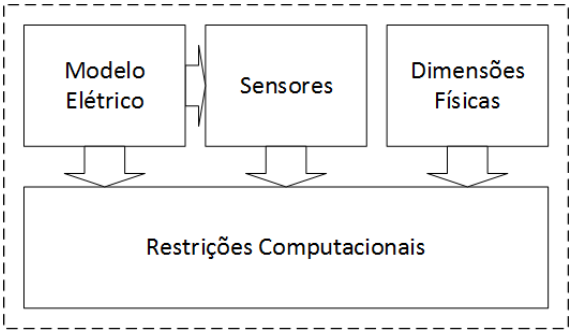


Figura 3.3: Estrutura do modelo do *quadcopter*

Através desta informação disponível no modelo elétrico, é possível calcular os dados de saída de um VANT real, assim como as suas restrições e limites. Um exemplo simples para um melhor entendimento pode ser o cálculo do tempo máximo de voo com o uso máximo de todas as suas capacidades. Existem algumas calculadoras online que permitem calcular os valores de alguns componentes, algumas até mesmo para configurar *quadcopters* que são utilizadas por comunidades de desenvolvedores de VANTs. No entanto, efetuando alguns testes com o mesmo conjunto de dados obteve-se resultados diferentes. Isto deve-se à falta de precisão das fórmulas existentes e ao facto de não existir um local específico onde se possa reunir informação sobre este tipo de aplicações de *quadcopters*.

Mediante esta situação procedeu-se à análise do funcionamento de um *quadcopter*, desde a sua constituição aos parâmetros que podem ser definidos. Estes dados de entrada encontram-se apresentados na Figura 3.4 estando agrupados pelos cinco tipos de componentes que constituem um *quadcopter* e por mais um conjunto de dados genéricos. Grande parte da informação é recolhida das fichas técnicas dos componentes e a maioria das calculadoras já contêm bases de dados com alguns dos componentes mais utilizados no mercado, tais como os motores, hélices, baterias, controladores ESC, *etc*.

Geral

No conjunto de dados mais gerais são necessárias algumas informações que são importantes ou que podem influenciar as restantes fórmulas. O primeiro campo está relacionado com o nível de refrigeração dos motores do *quadcopter* e encontra-se definido em cinco níveis desde o muito fraco ao excelente. De seguida é indicado o número de motores que a configuração do *quadcopter* contém, é de referir que este número é um dos fatores que vai influenciar o cálculo da potência que o *quadcopter* vai consumir. Outro facto importante é que só está a ser considerado um motor por cada extremo.

Outro campo necessário é o peso do conjunto completo do *quadcopter*, este pode ser indicado com o peso incluído dos componentes ou sem ele. Desta forma, garante-se que se está a contabilizar o peso total uma vez que existe sempre uma margem de erro medindo o peso de todos os componentes em separado. Os restantes campos necessários estão relacionados com fatores

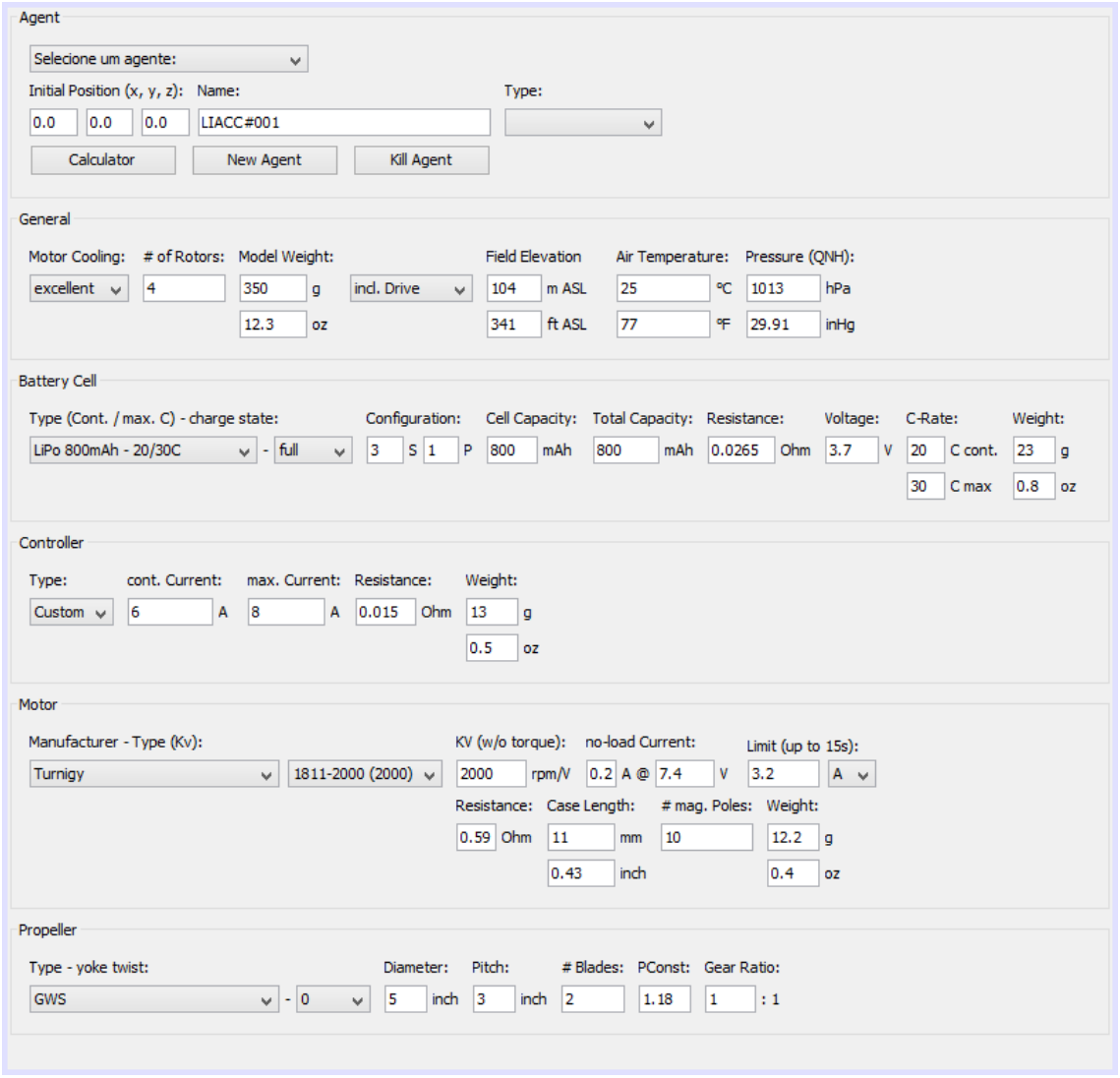


Figura 3.4: Parâmetros de configuração

de ambiente como a elevação acima do nível médio do mar, a temperatura e respetiva pressão atmosférica.

Bateria

Em relação ao conjunto das células da bateria, o primeiro conjunto de informações diz respeito à configuração da bateria, ou seja, quantas células tem em série e em paralelo. A partir desta informação e em conjunto com a capacidade e voltagem de cada célula, consegue-se calcular a capacidade total da bateria com a Equação 3.1 assim como a tensão nominal com a Equação 3.2.

$$TotalCapacity(A) = inBP * inBCellCap \tag{3.1}$$

$$RatedVoltage(V) = inBS * inBVcell \tag{3.2}$$

$$Load(C) = \frac{inGRotors * outMCurrent}{\frac{inBCap}{1000}} \tag{3.3}$$

A capacidade total da bateria e a saída máxima de corrente elétrica pode determinar o tempo de voo na pior das hipóteses e é calculado com a Equação 3.4. O resultado deste cálculo pode ser vantajoso e útil tanto para o utilizador do VANT como também para saber os limites que o VANT simulado pode atingir em tempo de voo à velocidade máxima antes de ficar sem energia. Este valor é meramente teórico e tem em conta apenas a capacidade da bateria, os resultados na realidade vão depender de aspetos como temperatura de ambiente, eficiência dos motores, resistências dos componentes e ligações, *etc*.

$$FlightTime(min) = \frac{\frac{inBCap}{outTDCurrentMax}}{1000} * 60 \tag{3.4}$$

À semelhança do cálculo do tempo e de voo máximo, com essas percentagens mencionadas podem ser calculados outros modos de voo como, por exemplo, voos agressivos ou controlados. Na Equação 3.5 é calculado o tempo de voo a pairar, este é baseado no uso de 85% da capacidade da bateria e na corrente consumida pelo *quadcopter* a pairar.

$$HoverFlightTime(min) = \frac{\frac{inBCap*0,85}{1000}}{outTDCurrentHover} * 60 \tag{3.5}$$

Nas Equações 3.6 e 3.7 são apresentadas as fórmulas que indicam o tempo da bateria em modo de escalada e voo controlado. Ambos são calculados da mesma forma com a capacidade da bateria e com 75% e 50% da corrente máxima do *quadcopter* respectivamente.

$$Climbout(min) = \frac{(\frac{inBCap}{outTDCurrentMax*0,75}) * 60}{1000} \tag{3.6}$$

$$Cruise(min) = \frac{(\frac{inBCap}{outTDCurrentMax*0,50}) * 60}{1000} \tag{3.7}$$

Por forma a se conseguir aumentar um pouco mais o tempo de voo e por forma aos cálculos representarem com mais realidade o uso da bateria, é comum calcular-se o tempo de voo a pairar com o uso de uma média de aceleração na ordem dos 60% como é apresentado na Equação 3.8.

$$Hoverw/AvgThrottle(min) = (outBHoverFlightTime * \frac{inAvgThrottle}{100}) + outBHoverFlightTime \tag{3.8}$$

Por fim o cálculo do peso da bateria é baseado no peso de uma célula multiplicado pelo número de células em série e por fim multiplicado pelo conjunto de células em paralelo como apresentado

na Equação 3.9.

$$Weight(g) = (inBS * inBWeight) * inBP \tag{3.9}$$

Controlador ESC

Quanto aos controladores ESC, estes são os componentes que permitem alimentar os motores com as baterias LiPo sem que estes sejam danificados ou até mesmo a própria bateria. Estes devem ser utilizados sempre de forma a que consigam fornecer a corrente necessária, pelo que se deve calcular a corrente máxima que o motor consegue consumir a partir da Equação 3.10.

$$MotorMaxDraw(A) = \frac{outMMeletricPower}{outBRatedVoltage} \tag{3.10}$$

Conhecendo-se a corrente máxima necessária dos motores, deve ser escolhido um controlador ESC que aguente essa corrente no mínimo e ainda acrescentar uma margem de erro de 2% como utilizado na Equação 3.11 para não correr o risco de danificar o componente logo na primeira utilização.

$$MinimumESC(A) = \frac{outMMeletricPower}{outBRatedVoltage} * 1,02 \tag{3.11}$$

O recomendado pelos padrões elétricos é a utilização de um controlador que tenha uma margem de segurança mínima recomendada de 10%. Mesmo que não existam controladores com o valor da corrente recomendada calculada deve-se sempre utilizar um com uma margem igual ou superior. A Equação 3.12 é semelhante a anterior com a diferença da percentagem da margem de segurança.

$$RecommendedESC(A) = \frac{outMMeletricPower}{outBRatedVoltage} * 1,10 \tag{3.12}$$

Motor

Em relação aos campos necessários para os cálculos nos motores, é necessário saber o Kv, ou seja, o número de *RPMs* por Volt, a corrente e voltagem sem carga e a resistência interna do mesmo. Outro campo que pode ser bastante útil é a corrente que o motor aguenta com um impulso adicional durante alguns segundos. A potência elétrica é calculada pela Equação 3.13 e é utilizada essencialmente para os cálculos da bateria. A potência mecânica é apresentada pela Equação 3.14 e já tem em conta a resistência interna do motor.

$$electricPower(W) = outMCurrent * outMVoltage \tag{3.13}$$

$$mech.Power(W) = (outMVoltage - outMCurrent * inMRi) * (outMCurrent - inMIo) \tag{3.14}$$

Através das duas potências previamente calculadas, pode-se descobrir assim se a eficiência do motor é boa o suficiente para que não seja necessário substituir os motores por outros com melhor desempenho. A eficiência é dada pela divisão da potência mecânica pela elétrica como apresentado na Equação 3.15.

$$Efficiency(\%) = \frac{outMmechPower}{outMelectricPower} * 100 \tag{3.15}$$

As rotações por minuto do motor sem carga são baseadas na constante Kv, em que 1 RPM por volt é baseado no inverso de um Kv de energia produzida por cada rotação do motor. Multiplicando a voltagem pelo Kv do motor sabemos então as RPMs do motor como apresentado na Equação 3.16 do ponto de vista teórico. Mas como já foi referido anteriormente na realidade este resultado pode ainda variar de acordo com fatores como as resistências da cablagem, modulação dos pulsos do controlador ESC, descarga da bateria, *etc*.

$$MotorNo-LoadRPM(rpm) = outBRatedVOltage * inMKv \tag{3.16}$$

Hélice

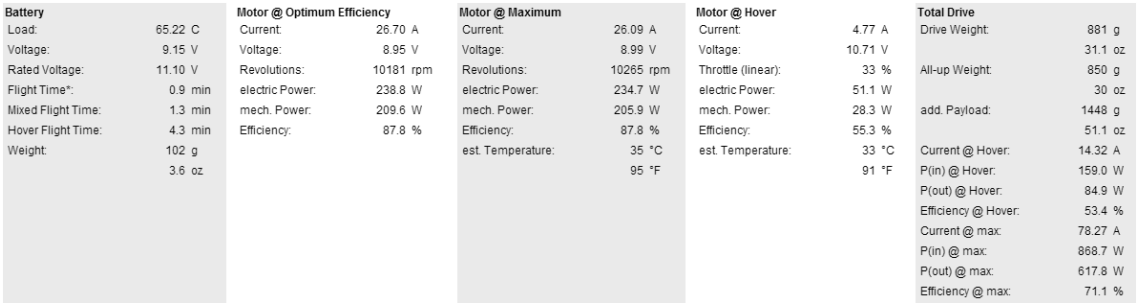
As hélices são os componentes que convertem o torque gerado pelos motores em força de elevação, e como são uns componentes que sofrem da ação de várias forças tornando-se assim complicado prever a força produzida por cada hélice com um elevado nível de precisão. Isto deve-se principalmente aos fatores como o diâmetro, o *pitch*, as formas, a área, o nível de flexibilidade do material e portanto os resultados obtidos incluem um certo nível de imprecisão. As forças produzidas através das rotações por minuto das hélices, estão também relacionadas com a densidade do ar. *Pitch Speed* apresentado na Equação 3.17 é a distância que se avança em uma rotação sem desvio multiplicado pelas RPMs, o que resulta na velocidade que o VANT vai atingir se não existir nenhum desvio.

$$StaticPitchSpeed = RPM * Pitch \tag{3.17}$$

Por forma ao voo ser fácil e estável, a força total gerada deve ser pelo menos duas vezes o peso do equipamento para que se tenha uma boa margem de manobra para executar os movimentos. Em relação à potência da hélice, quanto mais força gerar menos velocidade se vai conseguir como é apresentado na Equação 3.18.

$$Prop'sOutputPower = Thrust * PitchSpeed \tag{3.18}$$

Assumindo por exemplo que o diâmetro é maior que o *pitch*, isto vai resultar em mais força e menos velocidade máxima. Também se sabe que uma hélice pequena requer mais potência para gerar a mesma força do que com uma hélice maior. Dependendo da situação que se esteja a analisar pode-se estimar a potência necessária se for conhecido o *pitch speed* e a força que se precisa. Nas Equações 3.19 e 3.20 são apresentados os cálculos do *pitch speed* real com o desvio das hélices incluído e a velocidade do modelo real com o desvio das hélices mais o desvio do



Battery		
Load:	65.22	C
Voltage:	9.15	V
Rated Voltage:	11.10	V
Flight Time*:	0.9	min
Mixed Flight Time:	1.3	min
Hover Flight Time:	4.3	min
Weight:	102	g
	3.6	oz

Motor @ Optimum Efficiency		
Current:	26.70	A
Voltage:	8.95	V
Revolutions:	10181	rpm
electric Power:	238.8	W
mech. Power:	209.6	W
Efficiency:	87.8	%

Motor @ Maximum		
Current:	26.09	A
Voltage:	8.99	V
Revolutions:	10265	rpm
electric Power:	234.7	W
mech. Power:	205.9	W
Efficiency:	87.8	%
est. Temperature:	35	°C
	95	°F

Motor @ Hover		
Current:	4.77	A
Voltage:	10.71	V
Throttle (linear):	33	%
electric Power:	51.1	W
mech. Power:	28.3	W
Efficiency:	55.3	%
est. Temperature:	33	°C
	91	°F

Total Drive		
Drive Weight:	881	g
	31.1	oz
All-up Weight:	850	g
	30	oz
add. Payload:	1448	g
	51.1	oz
Current @ Hover:	14.32	A
P(in) @ Hover:	159.0	W
P(out) @ Hover:	84.9	W
Efficiency @ Hover:	53.4	%
Current @ max:	78.27	A
P(in) @ max:	868.7	W
P(out) @ max:	617.8	W
Efficiency @ max:	71.1	%

Figura 3.5: Resultados obtidos

modelo respetivamente. O desvio da hélice já se encontra numa forma reduzida da fórmula mas por norma é calculado a 20% e o desvio do modelo a 10%.

$$RealPitchSpeed + PropSlip(mph) = \frac{RPMs * inPPitch}{1320} \tag{3.19}$$

$$RealModelSpeed + PropSlip + ModelDrag(mph) = \frac{RPMs * inPPitch}{1509} \tag{3.20}$$

Após selecionados os parâmetros necessários, podem-se então efetuar os cálculos da potência da bateria e o tempo de voo, tensões e correntes dos motores assim como as rotações, entre outros dados que se poderão obter. Um conjunto de resultados de exemplo pode ser observado na Figura 3.5, com os dados calculados ainda existe a possibilidade de se gerarem gráficos, de onde se podem retirar mais algumas informações em relação à configuração do *quadcopter* para fins de otimização de corrente.

Estes conjuntos de dados encontram-se armazenados na estrutura de classes definida para o *quadcopter* apresentado na Figura C.4. A classe *Quadcopter* expande da classe *AbstractQuadcopter* que implementa todos os dados necessários em todos os tipos de *quadcopter*. Esta por sua vez implementa a interface que inclui os métodos utilizados pelos VANTs, por exemplo, velocidade, orientação, posição, *etc*. Voltando à classe *Quadcopter* esta é composta por uma *Frame*, essa frame no caso de ser com uma configuração em X, vai ser composta por uma bateria e quatro motores, cada um com a sua hélice. Também vai ter um conjunto de controladores para os sensores e comunicação com o XBee. Essas classes principais contém outras variáveis para auxílio dos cálculos, interfaces, *etc*.

3.4.2 Modelo Gráfico

O modelo gráfico apresentado na Figura 3.6 do *quadcopter* encontra-se dividido em três componentes fundamentais, a representação do modelo 3D, as dimensões físicas, as restrições computacionais e por fim a integração dos agentes através do Jason, as mesmas vão ser descritas de seguida. Para a instanciação do *quadcopter* no ambiente virtual, este necessita de ter uma estrutura previamente definida. Na plataforma foi definido um nó que agrupa os quatro tipos de veículos

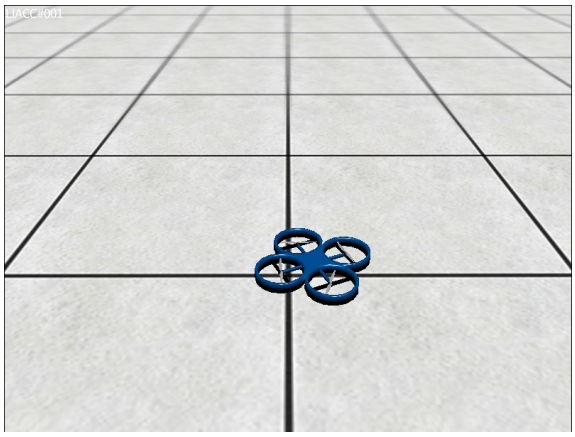


Figura 3.6: Quadcopter genérico

não tripulados, estes foram definidos como apresentado no Excerto de código 3.1 já a pensar em futuros desenvolvimentos. O modelo do *quadcopter* é incluído no *nodeUAVs* aquando da criação do agente, que vai criar a instância do agente dentro do Jason em simultâneo. Este procedimento vai ser descrito com mais detalhe posteriormente.

Dimensões Físicas

A dimensão física está representada através das suas características físicas e forças que podem ser geradas pelo modelo elétrico. Deve ser possível implementar as leis da física como a velocidade, aceleração, gravidade, para uma simulação mais dinâmica e realista do modelo. Para a construção do modelo de *quadcopter* é utilizado o formalismo de Newton-Euler que define três pressupostos:

- A estrutura do quadro deve ser rígida;
- A estrutura do quadro deve ser simétrica;
- O centro de massa do *quadcopter* coincide com o centro do quadro.

O momento de inércia é calculado assumindo que o *quadcopter* tem uma esfera central de raio r e massa m, dispondo de mais quatro massas organizadas em cruz representando os motores ligados ao quadro como se pode observar na Figura 3.7 [RG10].

Excerto de código 3.1 Nós dos Veículos

```
1 public void initVehiclesNodes() {
2     nodeVehicles.attachChild(nodeUAVs);
3     nodeVehicles.attachChild(nodeUGVs);
4     nodeVehicles.attachChild(nodeUSVs);
5     nodeVehicles.attachChild(nodeUUVs);
6     rootNode.attachChild(nodeVehicles);
7 }
```

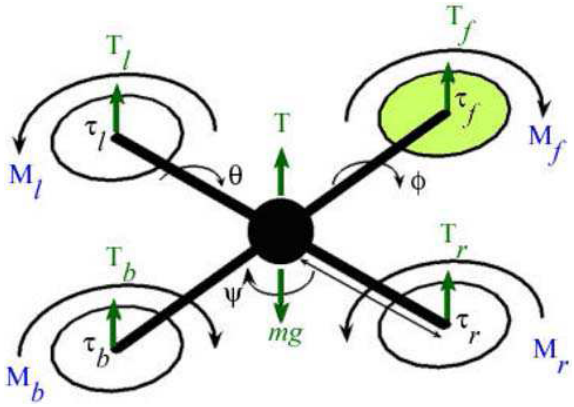


Figura 3.7: Diagrama conceptual de um *quadcopter* [RG10]

Tal como é ilustrado pelo diagrama conceptual na Figura 3.7, a altitude do *quadcopter* é controlada pela variação da velocidade de rotação de cada um dos motores. O motor da frente (Mf) e motor de trás (Mb) giram no sentido horário, enquanto o motor da direita (Mr) e o motor da esquerda (Ml) giram no sentido anti-horário. Esta configuração permite equilibrar as forças criadas por cada um dos pares de motores e assim permitir ao *quadcopter* manter-se estável durante o voo. Estes *quadcopters* são conhecidos pelas suas capacidades de manobra e contam com quatro movimentos simples alterando apenas as rotações de cada um dos motores de formas específicas.

Com o modelo de cinemática e dinâmica, é possível definir as forças e torques que atuam sobre o *quadcopter*. As forças incluem a sustentação aerodinâmica gerada por cada motor e a força gravitacional definida pela constante gravitacional g agindo contra o impulso total das forças geradas pelos quatro motores.

$$W_F b = \begin{bmatrix} -Mgs\theta \\ Mgc\theta\ s\phi \\ Mgc\theta\ c\phi \end{bmatrix} \tag{3.21}$$

Os momentos gerados são os torques que permitem efetuar os movimentos *Roll*, *Pitch* e *Yaw*. Na Figura 3.8, estão representados os torques que vão ser listados de seguida e encontram-se identificados pelas letras de (a) a (d) [RG10].

- ***Thrust*:** O impulso total do *quadcopter* é dado pela soma do impulso produzido por cada hélice, como se pode observar na Figura 3.8 (a).

$$T = T_f + T_r + T_b + T_l \tag{3.22}$$

- ***Rolling Torque*:** Este é o torque produzido pelo aumento do impulso do motor do lado esquerdo diminuindo ao mesmo tempo o impulso do motor do lado direito, ou vice-versa,

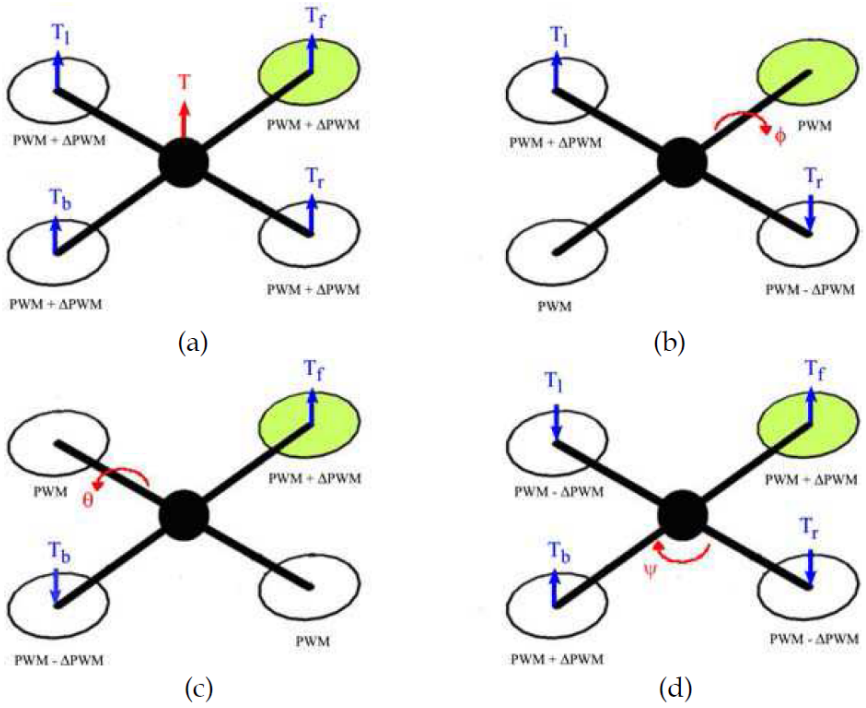


Figura 3.8: Forças e Torque de um *quadcopter* [RG10]

conforme mostrado na Figura 3.8 (b).

$$\tau_\phi = l(T_l - T_r) \tag{3.23}$$

- ***Pitching Torque*:** Este torque mostrado na Figura 3.8 (c) é produzido através do aumento do impulso do motor da frente, enquanto que se diminui o impulso do motor traseiro, ou vice-versa.

$$\tau_\theta = l(T_l - T_r) \tag{3.24}$$

- ***Yawing Torque*:** Este torque na Figura 3.8 (d) é o resultado de todos os quatro torques individuais gerados pelo par dos motores da frente e do traseiro com a rotação no sentido horário, enquanto que os motores da esquerda e da direita giram no sentido anti-horário. Isto permite criar um desequilíbrio no eixo dos z, podendo desta forma o *quadcopter* girar nesse eixo.

$$\tau_\psi = \tau_f + \tau_b - \tau_r - \tau_l \tag{3.25}$$

Restrições Computacionais

As restrições computacionais que devem simular as mesmas condições do VANT real, por exem-

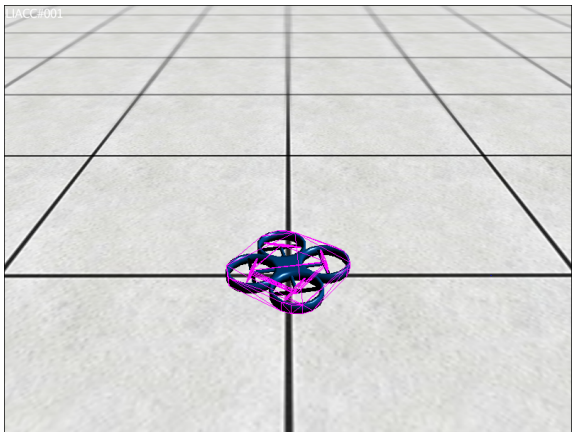


Figura 3.9: Quadcopter genérico com camada de colisões

plo, o VANT simulado deve seguir as velocidades calculadas a partir do modelo elétrico, que por sua vez indica a rotação dos motores e as respetivas forças que permitem que o VANT se mantenha em voo [SMD12].

O modelo virtual deve garantir que se o *quadcopter* estiver invertido, ou seja voltado para baixo o mesmo não se mantenha a voar uma vez que o impulso dos motores, o empurrariam contra o chão. Deve-se portanto garantir que determinadas ações não ocorram no ambiente simulado que ponham em causa a integridade física do equipamento real. Outra situação de exemplo seria a aplicação de um força externa sobre o *quadcopter*, este teria de reagir à força para se reposicionar na sua posição o mais rápido possível. Esta situação teria de ser corrigida através dos planos dos agentes, que mediante as suas perceções iriam corrigir a sua posição.

Para garantir a maior parte destas restrições foi aplicada uma grelha dinâmica através do método *CollisionShapeFactory.createDynamicMeshShape(quadcopter)*. Esta função cria uma forma dinâmica chamada *HullCollisionShapes* (com menos precisão) e é aplicada através de uma forma convexa que abrange todo o modelo 3D como se pode observar na Figura 3.9. No entanto esta foi apenas aplicada para diminuir o consumo do CPU, a forma mais correta para a plataforma seria utilizar o *GImpactCollisionShape* que é uma grelha com mais precisão para modelos dinâmicos 3D, só que ao contrário do que foi implementado este diminui a eficácia e aumenta o processamento. Este é utilizado para modelos dinâmicos complexos normalmente utilizado em realidade virtual ou simulações científicas.

Jason

O Jason permite a programação de comportamentos através de uma linguagem de alto nível, assim como a implementação de planos que podem ser interpretados pelo seu motor semântico em cada agente autónomo.

Para a execução dos projetos Jason será necessário fazer uma configuração prévia do projeto por forma a ser compilado e executado. Esta configuração é feita com um ficheiro com a extensão *mas2j* e é apresentada no Excerto de código 3.2. Neste deve-se indicar o tipo de infraestrutura

Excerto de código 3.2 Jason - Configuração do projeto

```
/* ALLMA - Agile inteLLigent autonoMous Aircraft */
MAS allma {
    infrastructure: Centralised
    environment: pt.up.fe.liacc.allma.jason.AllmaEnvironment
    classpath: "dist/lib/*.jar";
    aslSourcePath: "src/pt/up/fe/liacc/allma/jason/asl";
}
```

utilizada, na plataforma foi utilizado uma infraestrutura centralizada para agrupar os agentes, o que significa que estes só podem comunicar dentro do Jason. Caso se pretenda utilizar agentes distribuídos poder-se-ia utilizar uma infraestrutura descentralizada, como o JADE que integra este tipo de agentes.

Por forma a se ter acesso ao ambiente virtual, deve ser passado por parâmetro a classe que implementa a interface gráfica e o ambiente 3D. Ainda se deve indicar os diretórios das classes que são compiladas, assim como os diretórios de onde se vão encontrar os planos dos agentes por forma a não resultarem erros durante a compilação do projeto.

Em relação aos planos, pode ser utilizada a linguagem natural para as instruções e ações mas ainda é necessário conhecer a estrutura mínima que deve ser utilizada para representar os planos e perceções. No Excerto de código 3.3 é apresentado um plano de movimentos simples do *quad-*

Excerto de código 3.3 Jason - Plano de movimentos simples do *quadcopter*

```
// Initial Goal
!startGoal.

+!startGoal : true
   <- // .print("Initializing Agent:");
      // .print("Arm Vehicle");
      armVehicle(true);
      // .print("Move UP: 1.50m");
      moveUP(1.50);
      // .print("Move LEFT: 1.50m");
      moveLEFT(1.50);
      // .print("Move FORWARD: 1.50m");
      moveFORWARD(1.50);
      // .print("Move RIGHT: 1.50m");
      moveRIGHT(1.50);
      // .print("Move BACK: 1.50m");
      moveBACK(1.50);
      // .print("Move Down: 1.50m");
      moveDOWN(1.50);
      // .print("Disarm Vehicle");
      armVehicle(false).
```

Excerto de código 3.4 jME3 - Exemplo de *Internal Action*

```
1 @Action
2 public void moveFORWARD(Term[] terms) {
3     agent.moveFORWARD(getParamAsFloat(0, terms));
4 }
```

copter, o agente é instanciado com o objetivo que deve cumprir chamado *startGoal*. Para este objetivo o agente vai verificar que existe um plano e que este vai permitir atingir o objetivo, pelo que vai executar ação após ação à medida que estas vão sendo concluídas. Este plano permite ligar os motores do *quadcopter*, executar movimentos em todas as direções e voltar a aterrar no local de partida desligando por fim os motores.

Embora se possam escrever vários planos, quer de cooperação ou coordenação, é necessário que o ambiente 3D reconheça todos os movimentos e funcionalidades utilizadas. Estas ações utilizadas nos planos são identificadas na programação através da utilização de uma classe genérica que implementa a arquitetura do agente, ou seja, esta arquitetura integra todas as ações do agente assim como conhece todo o seu estado interno através das perceções gravadas.

Quando o Jason invoca uma ação no plano, através duma interface, são procurados as ações que correspondam ao nome e são chamadas de *internal actions*. No Excerto de código 3.4 é apresentado um exemplo deste tipo de ações. Quando é invocada a ação *moveFORWARD* é procurado este método identificado com a anotação *@Action* ou *@Action"moveFORWARD"*. Desta forma, consegue-se que o agente no ambiente virtual se desloque para a frente uma determinada distância indicada pelos parâmetros recebidos pelo array *Term[]*.

3.5 Ambiente Virtual dos Agentes VANT

Para a implementação do modelo físico do ambiente, foi utilizada uma plataforma denominada *Common ARTifact infrastructure for AGents Open environments* (CArtAgO) que vai permitir a implementação dos ambientes pré-definidos para os sistemas multiagentes de *quadcopters*. O CArtAgO encontra-se disponível através de uma API baseada em linguagem Java. Esta plataforma funciona com base no meta-modelo de agentes e artefactos, pelo que permite encapsular todos os objetos não autónomos do ambiente, permitindo aos agentes poderem explorá-los em tempo de execução. Deste modo, podem-se desenvolver e instanciar no ambiente artefactos que podem disponibilizar serviços aos agentes [RPV11]. Esta API também permite aos agentes a comunicação com serviços externos e traz uma melhor estruturação da interação entre agentes.

Voltando ao meta-modelo, este considera os agentes como entidades computacionais que pretendem realizar ou atingir um determinado objetivo. Os artefactos são como os recursos ou ferramentas que podem ser construídas, modificadas e partilhadas pelos agentes autónomos para atingirem os seus objetivos quer individuais, quer de grupo. Esta implementação é benéfica devido

às abstrações dos modelos físicos e são utilizadas como ferramentas ou recursos partilhados pelos agentes. Esta eficácia da abstração permite a criação de um modelo com interação mais detalhado graças aos recursos oferecidos pelos artefactos computacionais.

3.5.1 Estrutura do Ambiente Virtual

A estrutura interna do ambiente virtual encontra-se dividida em três nós principais, um que integra o terreno, um para a importação dos cenários e por último a integração do céu dinâmico. Como se pode observar no Excerto de código 3.5, todo o ambiente 3D do jME3 é implementado sobre uma estrutura em árvore que permite a integração dos objetos. Desta forma permite-se uma fácil organização quer do ambiente quer dos agentes como já foi apresentado no Excerto de código 3.1.

Endless Terrain

A Figura 3.10 apresenta uma captura de ecrã da implementação do *Endless Terrain* na plataforma de simulação. Este algoritmo utiliza o *TerrainGrid* que é uma extensão baseada nas ferramentas *TerraMonkey* no qual implementam algoritmos que garantem as rotinas para se visualizar terreno infinito no ambiente, por mais que um VANT se desloque no ambiente em qualquer sentido. Este algoritmo consiste em três componentes básicos, a classe *TerrainGrid*, e as interfaces *HeightMapGrid* e *TerrainGridListener*. A *TerrainGrid* é a classe central que trata desde o movimento da câmara, ao carregamento em memória das áreas de terreno e das respetivas notificações de alterações.

O *TerrainGridListener* necessita de verificar dois tipos de eventos, um deles é o *gridMoved(Vector3f)*. Este método calcula o novo centro do ambiente para a posterior atualização do terreno à sua volta, isto é, na atualização são adicionadas ou removidas as áreas necessárias. O segundo método é o *tileLoaded(Material material, Vector3f cell)* que trata da sua representação no

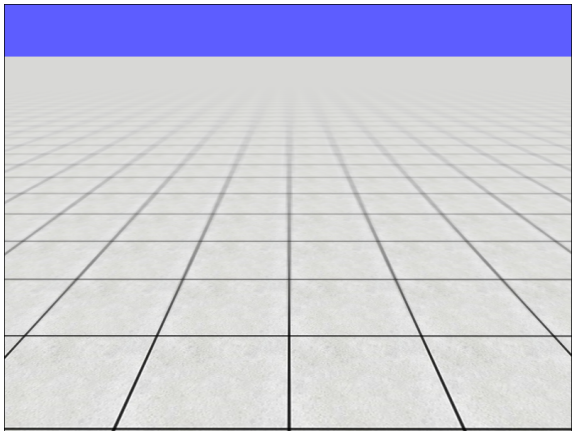


Figura 3.10: Terreno

Excerto de código 3.5 jME3 - Instanciação dos Nós do Ambiente

```
1 public void initEnvironmentNodes() {
2     nodeEnvironment.attachChild(nodeTerrain);
3     nodeEnvironment.attachChild(nodeCity);
4     nodeEnvironment.attachChild(nodeSky);
5     rootNode.attachChild(nodeEnvironment);
6 }
```

ambiente. Também podem ser adicionados outros *listeners* que permitem executar as mudanças no ambiente que se desejar enquanto são carregadas as áreas de terreno, por exemplo, podendo acrescentar outros objetos ao ambiente como edifícios, veículos, *etc*.

Em relação ao *HeightMapGrid* este serve para representar relevos no ambiente, através da leitura de um mapa passado por parâmetro baseado num mapa de 16 *bits*. Na plataforma para uma fase inicial este parâmetro é descartado e são utilizadas as definições por defeito que colocam os valores com elevação a zero. Este algoritmo foi utilizado pelo facto de os veículos numa fase inicial saírem dos limites do ambiente, desta forma não colidiam com o terreno e continuavam queda continua devido aos efeitos da física aplicados ao *quadcopter*. Sendo assim foi necessário implementar terreno contínuo por forma a este problema não se continuar a repetir.

Contudo a implementação desta solução traz as suas vantagens e desvantagens, ou seja, dependendo da configuração utilizada, quanto maior for o tamanho do terreno a ser apresentado mais memória será consumida na renderização dos limites do campo de visão da câmara. Para resolver este problema são utilizados vários níveis de cache que vão sendo invocados quando necessários.

Por forma a se conseguir otimizar o consumo de memória são utilizados filtros como *FractalHeightMapGrid* que introduzem ruído na paisagem mais distante do ambiente, permitindo diminuir o nível de detalhe. Isto tenta recriar o efeito de distância, quanto mais longe estiver de um

Excerto de código 3.6 jME3 - Instanciação do *Endless Terrain*

```
1  private void initTerrain() {
2      terrain = new TerrainGrid("terrain", 65, 257, ...);
3
4      terrain.setMaterial(mat_terrain);
5      nodeTerrain.attachChild(this.terrain);
6
7      TerrainLodControl c = new TerrainGridLodControl(terrain, getCamera());
8      c.setLodCalculator(new DistanceLodCalculator(257, 5.0f));
9      terrain.addControl(c);
10
11     floor_phy = new RigidBodyControl(0.0f);
12     terrain.addControl(floor_phy);
13     bulletAppState.getPhysicsSpace().add(floor_phy);
14 }
```

ponto, menos detalhe e menos percetíveis se tornam os objetos.

O modo como foi implementado encontra-se apresentado no Excerto de código 3.6 e funciona da seguinte forma. Primeiramente deve-se instanciar o objeto *TerrainGrid* indicando o nome e o tamanho da área e do bloco que vai ser apresentado. Internamente é gerada uma grelha que se encontra dividida em blocos. Esta é uma otimização para a própria apresentação das imagens que são visíveis. De seguida são definidos os materiais dos objetos que vão ser apresentados na grelha, caso não houvesse apresentação os objetos estariam implementados fisicamente, mas não visíveis para o utilizador. Feito isto, é necessário adicionar o objeto ao nó definido na estrutura para guardar o terreno, neste caso será o *nodeTerrain* que vai integrar a *TerrainGrid*.

Após isto foi adicionado um controlador do nível de detalhes do terreno, o *TerrainGridaLodControl* que se ajusta dependendo da proximidade ou distância que a câmara se encontra. Por fim, deve ser adicionada uma camada de colisões com a biblioteca da física para que os objetos consigam parar quando estiverem em contacto e serão adicionadas aos restantes objetos já inseridos no ambiente.

SkyDome

A Figura 3.11 apresenta a captura de ecrã do *SkyDome* implementado na plataforma de simulação. É um módulo disponibilizado nos repositórios oficiais do próprio jME3, onde já se incluiem funcionalidades adicionais para além da representação gráfica de um céu estático. Normalmente os céus estáticos são apresentados em pequenas aplicações que não necessitam de demasiados detalhes, e são implementados por um cubo que delimita os extremos do ambiente gráfico. Nesse cubo é aplicada uma textura, normalmente de uma fotografia relacionada com o tema do cenário. Entretanto nesta plataforma, pretende-se que o simulador represente algum detalhe da realidade e, como tal, o módulo utilizado permite recriar o céu dinamicamente numa cúpula e inclui funcionalidades como renderização dos movimentos das nuvens para que estas passem a ideia de profundidade.

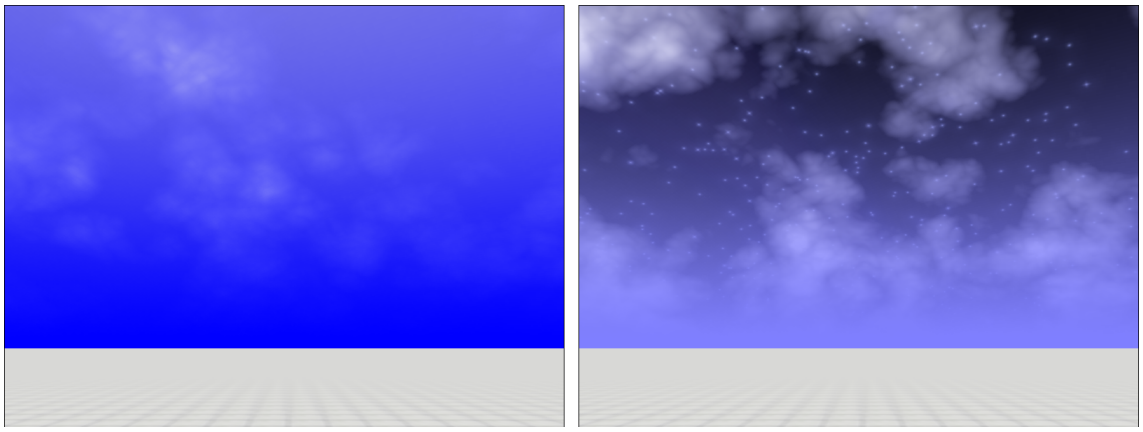


Figura 3.11: SkyDome

Esta também faz a transição entre o céu e o restante cenário através de um filtro de nevoeiro modificado, no qual não se sobrepõe aos restantes elementos gráficos apresentados. Para além disso, permite a separação do nevoeiro, das luzes de ambiente/sol, respetivas texturas e cores para o tempo durante o dia e durante a noite. Neste caso já estão incluídos controladores que tratam das transições do dia para a noite e vice-versa. Outros controladores tratam dos movimentos das nuvens, isto tudo pode ser configurado através de alguns parâmetros disponíveis para se alterar conforme as necessidades do utilizador. Os controladores também tratam da rotação das texturas da noite para o dia e da sua velocidade de transição através de um calendário de sistema próprio, atualizações da direção das luzes automaticamente entre outras funcionalidades, baseadas nos parâmetros que se quiser configurar. Por exemplo, num ambiente acelerado 1 segundo pode representar 1 minuto, 1 minuto igual a 1 hora e por adiante.

Por fim, este módulo ainda permite controlar o filtro de renderização das sombras, para que estas possam serem representadas corretamente de acordo com o horário do sistema. Foi implementado na plataforma através da função *initSky()* apresentado no Excerto de código 3.7 onde cada instrução se encontra comentada e corresponde às funcionalidades previamente referidas.

Excerto de código 3.7 jME3 - Instanciação do *SkyDome*

```
1  public void initSky() {
2      SkyDome skyDome = new SkyDome(assetManager, cam);
3      nodeSky.setQueueBucket(RenderQueue.Bucket.Sky);
4      nodeSky.addControl(skyDome);
5      nodeSky.setCullHint(Spatial.CullHint.Never);
6
7      // Enable the control to modify the fog filter.
8      FogFilter fog = new FogFilter(FOG_MODE.LINEAR, ColorRGBA.Blue, 0.5f, 0.5f,
           10f);
9      skyDome.setFogFilter(fog, viewPort);
10     skyDome.setControlFog(true);
11
12     // Add the directional light you use for sun.
13     skyDome.setSun(sun);
14
15     // Enable the control to modify your sunlight.
16     skyDome.setControlSun(true);
17     skyDome.setUseCalendar(true);
18
19     // Enable the control.
20     skyDome.setEnabled(true);
21     skyDome.initializeCalendar(1, 1, 24, 7, 4, 24);
22
23     // Add the SkyDome to the respective node.
24     rootNode.attachChild(nodeSky);
25 }
```

3.5.2 Importação de cenários

Em relação aos cenários utilizados no ambiente 3D da plataforma, podem ser importados através de modelos 3D do Blender. Os mesmos são importados para a estrutura do jME3 e todos os ficheiros de texturas são copiados para a pasta dos recursos. Desta forma podem ser utilizados modelos 3D mais realistas do *quadcopter* caso existam, na Figura 3.6 foi importado um modelo de um *quadcopter* genérico desenhado no Blender para representar o *quadcopter* real. No caso do modelo da cidade utilizou-se outro software de modelação 3D para gerar uma cidade aleatória com um ambiente simples para os *quadcopters* executarem algumas missões. Foi utilizado o Autodesk 3ds Max com um módulo externo, o ghostTown 0.317 que permite modelar uma cidade configurando parâmetros como o tamanho, quantidade de edifícios, espaços livres, *etc*. Na Figura 3.12 pode-se observar parte do modelo importado no lado esquerdo da imagem.

Para que o ambiente seja realista, todos os objetos que forem importados para dentro do nó *nodeCity* vão estar envolvidos numa camada de colisões estática. Esta camada é implementada com a classe *MeshCollisionShape* e é uma grelha própria para objetos estáticos. Esta permite envolver formas complexas, com aberturas e outros detalhes, tem a desvantagem de não permitir a deteção contra outras formas do mesmo tipo, o que não se aplica na forma como foi estruturada a plataforma. Para fins de importação é importante referir que o simulador está a implementar o ambiente 3D com os valores por defeito, o que quer dizer que na escala do simulador uma unidade de comprimento (1.0f) será igual a um metro, assim como o peso, em que uma unidade de peso (1.0f) será igual a um quilograma.

3.6 HILT (*HW-in-the-loop*)

3.6.1 Sensores

Os sensores devem emular o funcionamento real dos sensores e portanto ter uma componente capaz de representar o ruído existente no mundo real para cada tipo de sensor. Os sensores que

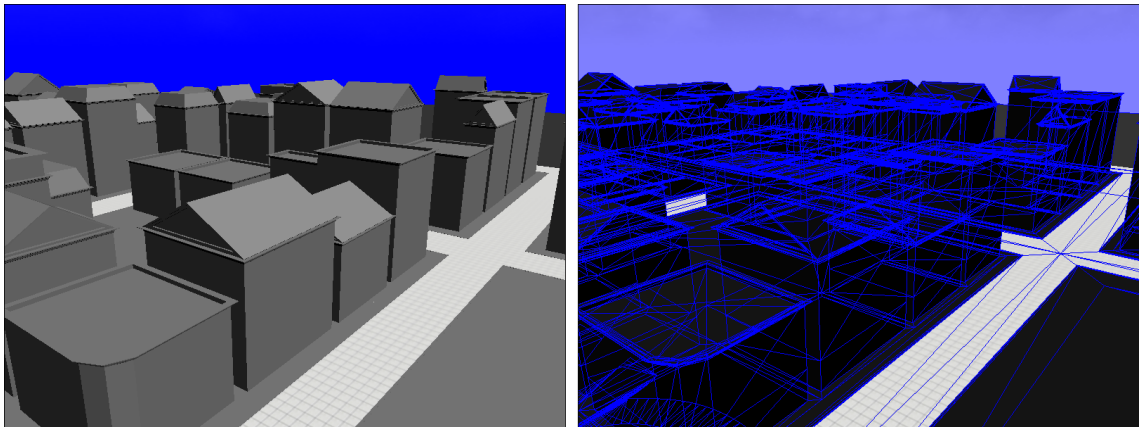


Figura 3.12: Cidade

foram implementados são:

Infravermelhos

Os sensores de infravermelhos possuem um emissor e um recetor que permite calcular a distância dos objetos, esse valor é obtido através de um fórmula característica disponibilizada pelo fabricante que permite converter a voltagem do sinal analógico numa distância. A imprecisão dos valores é obtida em determinados ângulos das superfícies e em determinados tipos de superfícies onde os sensores retornam alguns valores com erros no que diz respeito aos sensores de infravermelhos e sonares. Para se simular este funcionamento é utilizado um método do jME3, o *RayCast*, este permite uma fácil representação destes tipos de sensores, pois permitem retornar um conjunto de valores de colisões detetadas em linha reta no plano horizontal mediante uma orientação. O Excerto de código 3.8 apresenta a implementação do sensor no *quadcopter*.

Deve-se instanciar uma variável para os resultados das colisões e de seguida instanciar o *RayCast* na posição e orientação do *quadcopter*. Após isto, pode-se definir um limite máximo da distância que o sensor vai detetar as colisões, o uso deste método já contém alguma imprecisão no retorno dos resultados. Isto deve-se em grande parte ao nível de precisão definido no motor da física que resulta em alguns picos nos valores obtidos. Por fim pode-se indicar em que objetos vão ser detetadas as colisões e depois tratar os resultados da forma desejada, na implementação foi utilizado o nó principal que inclui todos os objetos estáticos e dinâmicos do ambiente 3D.

Sonares

Ao contrário dos infravermelhos que usam luzes para obter as distâncias, os sonares utilizam o som e funcionam enviando sinais e esperam o eco dos sinais para conseguirem calcular a distância. Estes têm uma forma que abrange mais área na deteção de colisões e são em forma de cone embora as curvas características dependem também dos fabricantes. A implementação do sonar vai ser na vertical para detetar a altitude do *quadcopter* e é utilizada a mesma base dos infravermelhos para os sonares. No Excerto de código 3.9 é apresentada a implementação das colisões com uma forma

Excerto de código 3.8 jME3 - RayCast

```
1  CollisionResults results = new CollisionResults();
2  Ray ray = new Ray(quadcopter.getLocalTranslation(), quadcopter.getLocalRotation
       ().multLocal(quadcopter.getLocalTranslation()));
3  ray.setLimit(1.4f);
4  rootNode.collideWith(ray, results);
5  if (results.size() > 0) {
6      CollisionResult closest = results.getClosestCollision();
7      if (closest.getDistance() > 1.4f ||  closest.getDistance() <= 0.0f) {
8      } else {
9          System.out.printf("Sensor: \%.2f m\n", closest.getDistance());
10     }
11 }
```

Excerto de código 3.9 jME3 - *Cone Collision Shape*

```
1 ConeCollisionShape shape = new ConeCollisionShape(0.2f, 1.5f, PhysicsSpace.
      AXIS_Y);
2
3 sonar = new RigidBodyControl(shape, 2f);
4 quadcopter.addControl(sonar);
5
6 bulletAppState.getPhysicsSpace().add(sonar);
```

em cone. Esta é feita através da aplicação do *coneCollisionShape* com os parâmetros do raio de abertura, da profundidade e do eixo de orientação. Criada esta forma deve ser adicionada as deteções de colisões e de seguida acrescentar o controlador no *quadcopter* virtual.

Acelerómetros, Giroscópios e GPS

O acelerómetro permite indicar em que sentido o *quadcopter* se encontra a deslocar, assim como o giroscópio indica a inclinação do mesmo nos três eixos. Tanto o acelerómetro como o giroscópio tem erro associado devido às vibrações do equipamento, na plataforma estes valores são mais precisos uma vez que o modelo tem toda a informação do seu estado.

Temos também o sensor GPS que é fundamental para o VANT conhecer a sua localização por forma a ser capaz de se mover no mundo real ou mundo virtual, este também tem um erro associado de precisão onde só permite localizar um objeto com um erro máximo de três a cinco metros. Sem esquecer também que este tipo de sensor está dependente do sistema de satélites de localização pelo que é sempre necessário ter o sensor com acesso direto com o exterior. Na maioria das vezes passando com estes equipamentos debaixo de infraestruturas perdem o sinal necessário dos três satélites para conseguirem calcular a sua posição. Na plataforma de simulação à semelhança dos acelerómetros e giroscópios estes são representados pela posição do modelo do *quadcopter* disponibilizado pela classe *Spatial* que contém toda a informação do modelo.

3.6.2 Protocolo de comunicação

A implementação de uma vasta variedade de modelos de *quadcopters* significa que o sistema (neste caso o simulador) deve ter a capacidade de integrar vários modelos e que estes sejam capazes de interagir e comunicar simultaneamente no mesmo mundo. Esta meta permitirá testar vários cenários com múltiplos VANTs sem ter gastos de recursos para fins de teste. Todos os modelos juntos devem responder com todas as informações que são geradas dinamicamente por VANTs como se fossem reais e devem ter o conhecimento de outros agentes no mesmo campo ao mesmo tempo. A informação deve ser consistente e concisa para assegurar que todos os agentes se encontram a observar os mesmos objetos.

A definição de coordenação apresenta-se como a unificação, integração, sincronização dos esforços das entidades para fornecer uma unidade de ação na busca de objetivos comuns. As

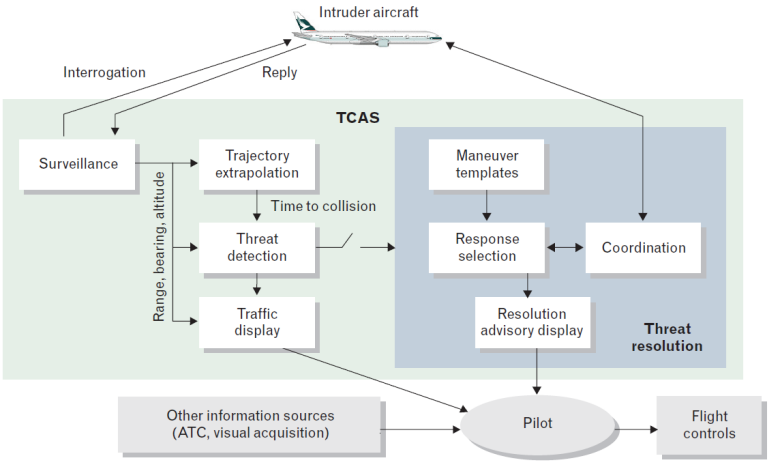


Figura 3.13: Sistemas Anti Colisão de Tráfego (SACT) [KD07]

estratégias para alcançar a coordenação encontram-se garantidas através de funções básicas de planeamento, organização, gestão de recursos, direção e controle dos VANTs. Um dos sistemas analisados para a coordenação dos VANTs e que garantem que se evitem colisões são os Sistemas Anti Colisão de Tráfego (SACT), na Figura 3.13 é apresentado o esquema de funcionamento destes sistemas. Estes sistemas estão organizados em vários elementos, em primeiro encontram-se os sensores que recolhem as informações sobre os VANTs. Essas informações podem ser a sua posição, velocidade, aceleração entre outras e são passadas para um conjunto de algoritmos que determinam se existe risco de colisão e por fim são comunicados os resultados aos VANTs.

Se um risco for identificado, um segundo conjunto de algoritmos determina a resposta mais apropriada para cada VANT. Se ambos os VANTs utilizarem sistemas SACT a resposta é coordenada para garantir as manobras numa trajetória compatível deste modo pode-se evitar as colisões [KD07].

Na Figura 3.14 é demonstrado um exemplo de cenário que se encontra neste tipo de sistemas, no qual o sistema SACT envia uma mensagem para um dos *quadcopters* subir e o outro recebe a mensagem para descer para uma rota mais baixa evitando assim a colisão eminente. Essas mensagens enviadas seguem a estrutura apresentada na Tabela 3.2.

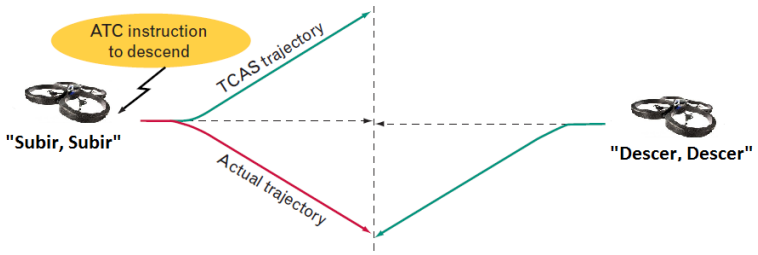


Figura 3.14: SACT - Cenário exemplo

Tabela 3.2: SACT - Mensagens e Avisos [oT11]

Tipo	Mensagem	Descrição
TA	Tráfego; Tráfego.	Aviso de VANT nas proximidades (contacto visual), preparação para efetuar manobras caso receba notificação de resolução.
RA	Subir; Subir.	VANT vai passar por baixo, começar a subir.
RA	Descer. Descer.	VANT vai passar acima, começar a descer.
RA	Aumentar subida.	VANT vai passar mesmo abaixo, subir mais.
RA	Aumentar descida.	VANT vai passar mesmo acima, descer mais.
RA	Reduzir subida.	VANT provavelmente muito abaixo, subir devagar.
RA	Reduzir descida.	VANT provavelmente muito acima, descer devagar.
RA	Subir; Subir imediatamente.	VANT que ia passar acima vai passar agora abaixo, mudar de descida para subida.
RA	Descer; Descer imediatamente.	VANT que ia passar abaixo vai passar agora acima, mudar de subida para descida.
RA	Manter velocidade vertical; Manter.	VANT evitado se manter altitude.
RA	Ajustar velocidade vertical; Ajustar.	VANT consideravelmente longe, estabilizar voo.
RA	Controlar velocidade vertical.	VANT acima ou abaixo, permanecer em voo nivelado.
RA	Cruzar.	Passando por um nível de outro VANT, proceder de acordo com RA.
CC	Livre de conflitos.	VANT deixou de ser considerado um conflito. Proceder com plano de voo.

3.6.3 Mecanismos para a simulação simbiótica

Conectar o mundo virtual com o mundo real significa implementar um meio de comunicação entre os dois mundos, bem como especificar como esta comunicação pode ser realizada. Esta ligação pode ser usada de várias formas: para a recolha de dados do mundo real e processamento de dados no mundo virtual, para enviar as ações de mundo virtual para o mundo real ou de um modo híbrido que permite a receção e envio de informação entre os dois mundos em simultâneo.

Como se pode observar na Figura 3.15, ambos os *quadcopters* devem conhecer e terem implementados o protocolo *UAVTalk* para se conseguir comunicar através dos módulos XBee. Este protocolo pode funcionar de forma síncrona ou assíncrona enviando e recebendo pacotes *UAVObject* que contém as informações relativas ao estado interno do *quadcopter* podendo ser enviadas em intervalos de tempo configuráveis. Essas informações sobre os parâmetros do objeto de navegação encontram-se definidas em XML e o analisador é que irá gerar o *UAVObject* necessário para ser enviado para os *quadcopters*. Este protocolo do UAVTalk no desenvolvimento do sistema físico acabou por ser simplificado através de uma camada adicional e assim permitiu-se simplificar a troca de mensagens entre o *quadcopter* real e virtual através dos módulos XBee. Foram definidas as estruturas dos pacotes, à semelhança das mensagens FIPA ACL permitindo enviar os dados essenciais. Desta forma reduz-se o consumo de energia do equipamento e gasta-se menos

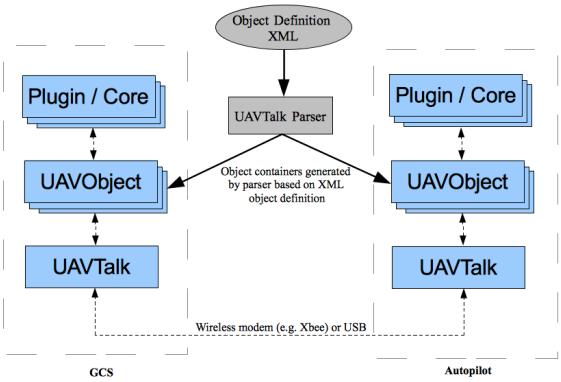


Figura 3.15: UAVTalk

tempo a analisar pacotes que podem vir a ser descartados.

XBee - Utilização da API

Para se trabalhar com os módulos de rádio XBee, foi utilizada uma API em Java, já existente e específica para a comunicação através dos módulos XBee/XBee-Pro versão 1 (802.15.4) e versão 2 (ZB/ZigBee Pro). O objetivo da utilização desta API é garantir a flexibilidade e a facilidade no uso da API para interagir diretamente com os módulos, não restringindo a implementações específicas mas sim para orientar na forma como devem ser utilizados. Esta API necessita de ter disponível no sistema operativo utilizado a versão igual ou superior do Java 5 e ainda o RXTX para funcionar corretamente. Estes modos podem ser configurados para trabalhar de duas formas, umas delas já foi referida a API e ainda existe uma segunda configuração a AT. Esta API requer que o módulo esteja configurado no modo API, definido com a configuração AP = 2. Esta configuração será descrita com mais detalhe na Secção A.2.3.

No Excerto de código 3.10 é apresentada a configuração inicial da API. A classe *XBee* é a interface de comunicação com o módulo XBee e permite toda a coordenação do envio e receção dos dados através de pacotes. Uma instância da classe *XBee* deve ser criada para permitir a comunicação através da porta de comunicação serial, ou usar um adaptador serial para USB uma vez que a grande parte dos computadores já não traz a porta série. Deve-se conectar o adaptador no local da plataforma com o XBee que estiver configurado como API. O método *open()* deve ser invocado para se estabelecer a comunicação com o rádio, deve-se indicar a porta COM no qual está conectado e a respetiva velocidade de comunicação. A porta série depende do sistema operativo, no Windows será COM1 ou a última porta COM virtual instalada. Em relação à velocidade de comunicação, por defeito é utilizado 9600 baud rate mas na plataforma é utilizado 57600 baud rate para obter uma maior velocidade de transmissão.

Existem dois métodos diferentes para a receção de pacotes, um deles é o *getResponse()* e o segundo é através de *PacketListener*. O primeiro método é apenas recomendado para aplicações com uma única tarefa em execução enquanto o segundo método é mais aconselhado para múltiplas

Excerto de código 3.10 XBee - Instanciação do *XBeeControl*

```
1  private XBee xbee;
2
3  public XBeeControl(String port) {
4      try {
5          xbee = new XBee();
6          xbee.open(port, 57600);
7      } catch (XBeeException ex) {
8          Logger.getLogger(XBeeControl.class.getName()).log(Level.SEVERE, null,
               ex);
9      }
10 }
```

tarefas em simultâneo. O Excerto de código 3.11 demonstra o funcionamento da receção de pacotes através do método *getResponse()*, no qual pode-se passar como parâmetro o tempo máximo de espera de cada pacote em milissegundos. Este método encontra-se dentro de um controlador, que é uma *thread* que está constantemente em atualização enquanto a plataforma estiver a ser executada. Se não for indicado o tempo de espera o processo vai bloquear enquanto não receber um pacote. Esta abordagem também só é possível utilizar caso se tenha só um *quadcopter* real para comunicar.

Excerto de código 3.11 XBee - Recepção de pacotes

```
1 XBeeResponse response = xbee.getResponse(1);
2 // Checks the type of the packet and discards if not match.
3 if (response.getApiId() == ApiId.ZNET_RX_RESPONSE) {
4     ZNetRxResponse rx = (ZNetRxResponse) response;
5     // Data from sensors, e.g., accelerometers, gyroscope.
6     String packetString = ByteUtils.toString(rx.getData());
```

Como não houve oportunidade de se comunicar com vários equipamentos, não foi possível testar a implementação do *PacketListener*, neste caso a abordagem seria instanciar uma classe XBee geral e desta fazer a distribuição de todos os pacotes para os respetivos modelos dos *quadcopters* virtuais através do tratamento dos pacotes efetuados no método *processResponse* apresentado no Excerto de código 3.12, é de referir que o processamento das mensagens é efetuado numa tarefa à parte.

No caso do envio de pacotes, a classe *XBee* disponibiliza dois métodos: *sendAsynchronous()* e *sendSynchronous()*. Como os próprios nomes dos métodos indicam, o método assíncrono apresentado no Excerto de código 3.13 envia o pacote e não aguarda nenhuma resposta, neste tipo de plataforma de simulação será a melhor forma a utilizar, uma vez que o ciclo de processamento não pode parar para enviar de novo os pacotes perdidos. O máximo de informação que pode enviar-se em cada pacote é de 255 bytes, pelo que se deve escolher sempre para enviar a informação mais

Excerto de código 3.12 XBee - *Packet Listener*

```
xbee.addPacketListener(new PacketListener() {
    public void processResponse(XBeeResponse response) {
        // Distribuir pacotes pelos modelos virtuais...
    }
});
```

importante. Se não se indicar o destino do pacote este é enviado em *broadcast* para todos os módulos XBee que o conseguirem receber.

Excerto de código 3.13 XBee - Envio de pacotes assíncronos

```
ZNetTxRequest sendPacket;
int[] sendData = {//Some Values};

// Configure packet with destination XBee Address and Data to Quadcopter.
sendPacket = new ZNetTxRequest(new XBeeAddress64(0x00, 0x13, 0xA2, 0x00, 0x40,
    0x31, 0xFB, 0xE3), sendData);

xbee.sendAsynchronous(sendPacket);
```

O método alternativo apresentado no Excerto de código 3.14 é mais aconselhado a ser utilizado no envio de ações para o *quadcopter* real no qual seja necessário uma resposta imediata para se saber se recebeu as instruções corretamente por exemplo. À semelhança da receção dos pacotes, também existe um tempo máximo de espera nos envios de pacotes síncronos.

Excerto de código 3.14 XBee - Envio de pacotes síncronos

```
ZNetTxRequest sendPacket;
int[] sendData = {//Some Values};

try {
    // Configure packet with destination XBee Address and Data.
    sendPacket = new ZNetTxRequest(new XBeeAddress64(0x00, 0x13, 0xA2, 0x00, 0
        x40, 0x31, 0xFB, 0xE3), sendData);

    XBeeResponse response = xbee.sendSynchronous(sendPacket, 1000);
} catch (XBeeTimeoutException e) {
    // No response was received in the allotted time
}
```

3.7 Conclusão

Neste capítulo procedeu-se à explicação de toda a implementação assim como também foram descritos todos os componentes que foram utilizados. Viu-se de que forma um *quadcopter* se pode mover com o objetivo de modelar os movimentos básicos característicos, demonstrando as fórmulas matemáticas das forças e torques gerados. Foi decidido avançar com a utilização da linguagem Java, pois permite uma maior interoperabilidade entre os vários sistemas operativos e também permite a fácil adaptação para o uso nas plataformas móveis. O ponto mais crítico na modelação está relacionado com os sensores, uma vez que estes devem ser bastante precisos, a fim de evitar danificar o equipamento real com tomadas de decisões baseadas em dados simulados do ambiente, mas através da simulação simbiótica este problema é eliminado de uma forma bastante eficaz.

Na Figura C.5 apresenta-se um fluxograma que pretende demonstrar todo o fluxo de dados dos componentes de toda a implementação desta plataforma, dando uma visão geral do fluxo de funcionamento. Foi desenvolvida uma plataforma de simulação, que permite aos investigadores testar e avaliar as respostas dos VANTs em cenários dinâmicos da vida real. O objetivo principal foi ser capaz de testar o controlo, a coordenação e comunicação dos VANTs de forma a que fosse capaz de evitar danificar os equipamentos físicos. Esta plataforma teria que, para além de suportar múltiplas entidades, deve funcionar de forma modular para que a sua extensão e respetivas manutenções se tornassem simples. Por último, os requisitos apresentados pela plataforma exigiam que fosse possível receber dados dos equipamentos reais para que se conseguisse corrigir e melhorar a sua eficácia, pelo que se implementou o sistema com os módulos XBee e com um protocolo de comunicação bastante simples para a troca de informação.

A utilização dos módulos como o *Endless Terrain*, *SkyDome* entre outros permitiu dar um pouco de realismo à própria plataforma e ao mesmo tempo otimizar a eficácia na renderização do ambiente 3D. Este projeto é apenas um início na co-simulação de *quadcopters* no que diz respeito a simulação do *quadcopter* e da simulação do funcionamento dos seus componentes fisicos. Mesmo quando todas as funcionalidades propostas estiverem concluídas, pode-se expandir ainda mais a ferramenta para dar suporte a outros sensores, tais como câmaras, microfones, luminosidade, *etc*.

Capítulo 4

Testes, Verificação e Validação

Neste capítulo descrevem-se os testes de funcionalidade e de desempenho da plataforma de simulação, por último, analisam-se os resultados obtidos.

4.1 Testes Selecionados

Para se analisar corretamente a arquitetura e a implementação da plataforma de simulação, várias podem ser as métricas (diretas ou indiretas) como são os testes que permitem verificar a integridade e desempenho de cada módulo do sistema.

Desta forma a escolha dos testes vão incidir sobre o módulo da simulação simbiótica, da co-simulação e dos agentes. Dado a grande parte do trabalho ter incidido na integração das várias tecnologias em conjunto e na otimização dos recursos que utilizam, as seguintes métricas foram aplicadas para os testes:

- Testes de Funcionalidade:
  – Teste de Simulação Simbiótica;
  – Teste de Co-Simulação de Infravermelhos;
  – Teste de Simulação de Agentes.

- Testes de Desempenho:
  – Teste de Escalabilidade da Plataforma;
  – Teste de Desempenho da Plataforma.

A execução e avaliação dos testes foram efetuadas em duas configurações de computadores com as especificações apresentadas na Tabela 4.1. Desta forma, deve ser considerada cada uma das configurações utilizadas na análise dos resultados obtidos.

Tabela 4.1: Especificação dos computadores de testes

PC	Home	Lab
CPU	Dual CPU T2410	i5-2400
Velocidade CPU	2.00GHz	3.10GHz
Memória RAM	3.00GB	8.00GB
Placa Gráfica	NVIDIA GeForce 9300M GS	AMD Radeon HD 6570
Sistema Operativo	Windows 8.1 Pro	Windows 7 Pro

4.2 Testes de Funcionalidade

As próximas seções vão descrever os testes realizados com a métrica de funcionalidade, desde a configuração dos testes, à obtenção e interpretação dos resultados.

4.2.1 Teste de Simulação Simbiótica

O primeiro cenário de testes da plataforma passa pela execução do teste de simulação simbiótica entre o *quadcopter* real e o respetivo avatar no ambiente virtual. Este vai ser composto pelo *quadcopter* real equipado com um módulo XBee apresentado no lado direito da Figura 4.1, no qual vai transmitir para a plataforma de simulação os dados do acelerómetro e do giroscópio.

O objetivo será que o próprio *quadcopter*, seja representado no ambiente virtual conforme o estado que se encontra na realidade. Neste teste foram apenas considerados os movimentos do *quadcopter* numa posição fixa para se ter mais precisão na observação dos movimentos. No lado esquerdo da Figura 4.1 encontra-se o *quadcopter* virtual com os mesmos ângulos em que o real se encontra.

Foram observados dois gráficos apresentados na Figura 4.2, na qual do lado esquerdo representa os dados do giroscópio no simulador e no lado direito o gráfico obtido diretamente pelo programa de controlo da placa OpenPilot CC3D que inclui o sensor do giroscópio.

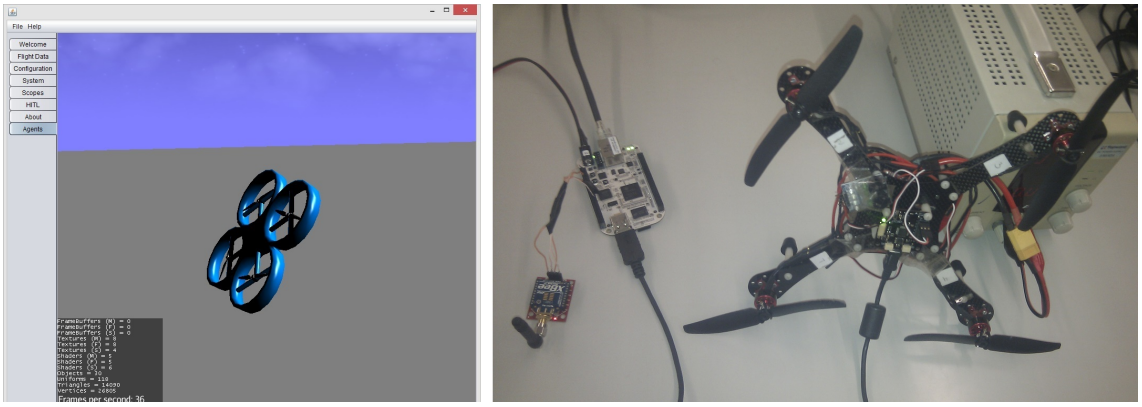


Figura 4.1: Configuração do Cenário: *Quadcopter* virtual(*avatar*) e real

O objetivo será verificar a similaridade dos valores do ambiente virtual com os dados obtidos pela ligação simbiótica. Os gráficos foram capturados com os dois sistemas ligados em simultâneo e com uma determinada posição fixa.

Apesar de não conseguir fazer uma captura dos resultados do real, em termos de dados numéricos pela observação dos gráficos, consegue-se verificar que o *quadcopter* virtual segue a mesma resposta dos dados recebidos pela ligação simbiótica. Isso pode-se verificar nos pequenos ajustes que são observados nas linhas (verde e vermelha) nas quais ajustam para os valores recebidos. Devido à velocidade de atualização no simulador o *quadcopter* faz os ajustes para estabilizar de acordo com as forças físicas até às atualizações seguintes.

De referir ainda que os eixos de altura e profundidade no ambiente virtual estão invertidos em comparação com os eixos do sensor real.

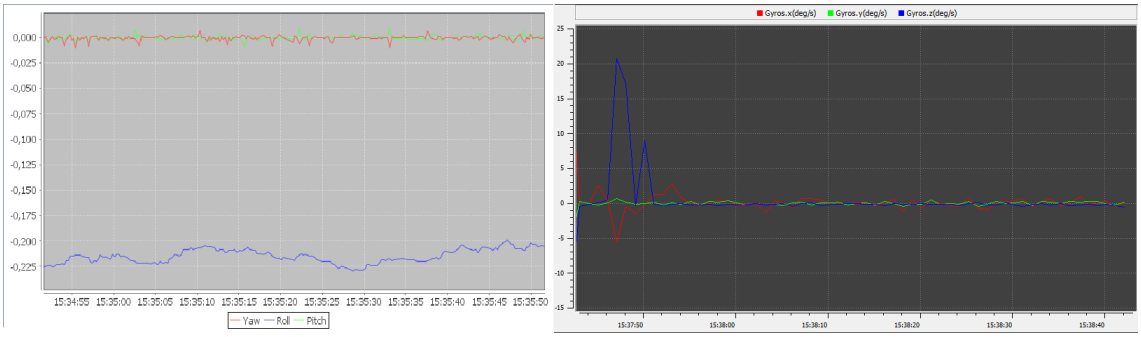


Figura 4.2: Resultados: Giroscópio *quadcopter* virtual e real

Estes foram observados em vários testes executados, desde a primeira troca de dados entre o equipamento real e o simulado, em alguns dos testes executados com a receção de pacotes em velocidades mais reduzidas apresentavam-se falhas na atualização pelo que no ambiente virtual notava-se umas falhas entre as atualizações, isto é, não passava a noção de um único movimento fluído. Devido a esta situação foram experimentadas várias velocidades de atualização até este problema não se verificar.

4.2.2 Teste de Co-Simulação de Infravermelhos

O segundo cenário de testes da plataforma passa pela comparação do funcionamento dos sensores de distância reais com os simulados, estes dados são obtidos pela mesma forma que são recebidos os dados do giroscópio por exemplo. Este vai ser composto pelo *quadcopter* real equipado com um módulo XBee no qual vai transmitir para a plataforma de simulação os dados do sensor de infravermelhos orientado para a frente do *quadcopter*.

O *quadcopter* real encontra-se numa estrutura quadrada com um metro e meio de lado, por forma a que se garanta que os sensores não saiam fora do nível de alcance máximo. À semelhança do ambiente real foi também inserido no ambiente virtual uma estrutura quadrada representando os limites máximos dos sensores. Na Figura 4.3 apresenta-se um exemplo da configuração deste cenário de teste.

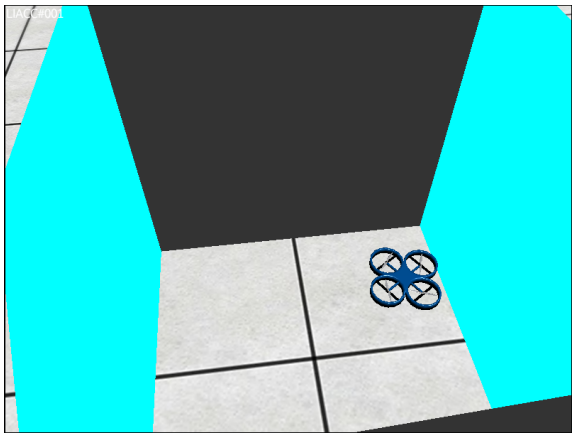


Figura 4.3: Configuração do Cenário: Quacopter e Estrutura limite

O objetivo é apresentar-se resultados bastantes próximos dos dados reais, uma vez que os sensores reais perdem precisão quanto maior for a distância. Neste teste foi apenas considerado a utilização do sensor de infravermelhos na frente do *quadcopter*.

Para a obtenção dos resultados a experiência passou por configurar um cenário de teste tanto no ambiente virtual como no ambiente real. Foram medidas várias distâncias e colocado o sensor de infravermelhos por forma a se retirar uma amostra de dados significativa. O mesmo procedimento se usou no ambiente virtual. Na Tabela 4.2 estão apresentados os dados processados que indicam as médias das distâncias obtidas e respetivas variâncias do erro.

Os resultados obtidos permitem verificar que o sensor de infravermelho real perde precisão conforme a distância aumenta do ponto onde foram colocados. Isto deve-se em grande parte ao erro incluído na curva da característica fornecida pelo fabricante e pode-se mesmo verificar que quanto maior a distância, mais aumenta a variância do erro da amostra.

Quanto aos valores obtidos dentro das mesmas distâncias no ambiente virtual, e com uma amostra dos resultados também significativa conseguem-se obter resultados mais precisos e a variância bastante menor uma vez que estes dados são calculados e não tem tantos fatores a influenciar os resultados como no equipamento real. Apesar disso o simulador integra também um

Tabela 4.2: Resultados - Sensores de Infravermelhos

Distância - Real	43cm	49cm	73cm	110cm
Infravermelhos Real				
Voltagem - Média (V)	1,402491672	1,274700115	0,863866967	0,610445238
Distância - Média (cm)	43,439977	48,21790724	73,77832211	106,5052079
Distância - Variância (cm)	0,249932827	0,223685504	1,124206641	3,597595932
Infravermelhos Virtual				
Voltagem - Média (V)	1,403517479	1,242453835	0,747637158	0,497719323
Distância - Média (cm)	43,4076384	49,5919969	73,4984921	110,131513
Distância - Variância (cm)	0,0055960	0,0058385	0,0037863	0,0032787

pequeno erro que se verifica estar relacionado com o nível escolhido da precisão no motor da física (*jBullet*).

A partir dos resultados obtidos pode-se concluir que os erros dos sensores reais podem ser melhorados por via da simulação, caso os *quadcopters* tenham conhecimento da espaço físico no ambiente virtual. Desta forma os *quadcopters* virtuais e reais têm acesso aos dados dos sensores virtuais que são mais precisos.

4.2.3 Teste de Simulação de Agentes

O terceiro cenário de testes da plataforma passa pela execução de um plano simples, ou seja, o *quadcopter* vai ser instanciado numa posição inicial do ambiente com um objetivo que deve cumprir. O motor de cognição do Jason vai procurar um plano que vai dar uma resposta ao objetivo com o qual foi instanciado e executará as ações que estiverem desenhadas no plano.

Este cenário vai ser composto pelo *quadcopter* virtual, que representa fisicamente o agente no ambiente virtual e por um objetivo principal chamado *start* e um plano que fornece uma resposta simples de exemplo.

O objetivo é o *quadcopter* executar as ações mediante o plano de ações disponível através do interpretador do Jason. Estes comandos presentes nos planos são convertidos no Jason para a linguagem de baixo nível para a atualização do ambiente virtual. Neste teste foi apenas considerado a utilização de um plano que integra movimentos simples como deslocar o *quadcopter* em vários sentidos e voltar ao local de partida no fim.

Na Figura 4.4 apresenta-se uma captura de ecrã da plataforma no qual são apresentados as coordenadas que o agente executou através do plano que lhe permite atingir o seu objetivo inicial.

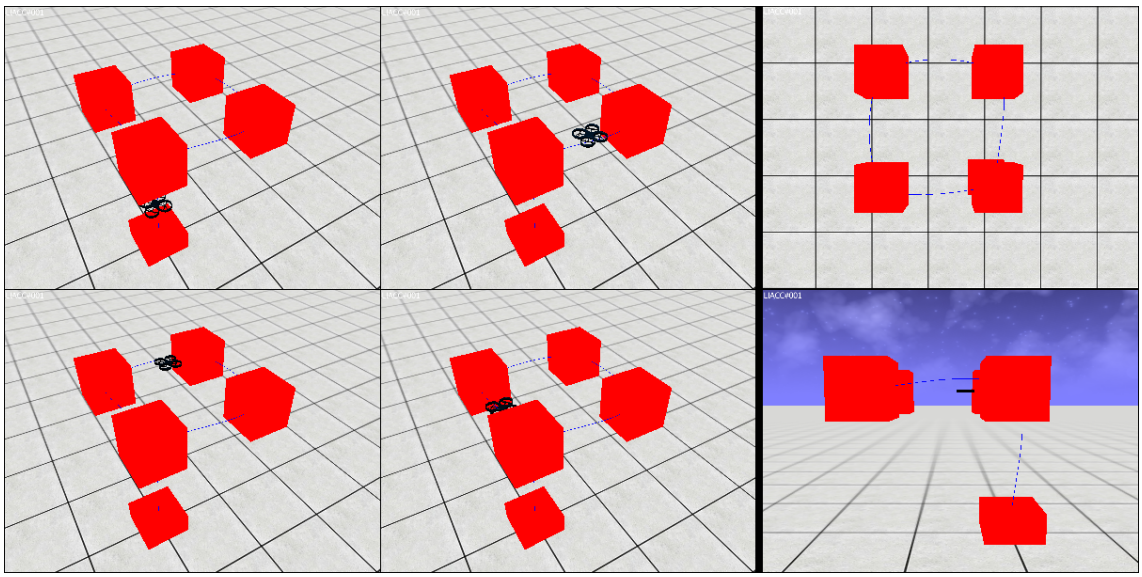


Figura 4.4: Resultados: Execução de Plano

Neste caso foram apresentadas cinco caixas a vermelho que representam os pontos por onde o *quadcopter* deve passar, a linha azul representa o caminho que deve seguir

Nas quatro primeiras imagens do lado esquerdo apresenta-se um *quadcopter* virtual a seguir as retas e a passar pelas coordenadas indicadas no plano, as duas imagens do lado direito apresentam uma vista superior e lateral do cenário de teste em que se pode observar a coordenada do *quadcopter* inicial se encontra no solo e as restantes a 1.5m de altura de acordo com o plano apresentado no Excerto de código 3.3.

4.3 Testes de Desempenho

As próximas seções vão descrever os testes realizados com a métrica de desempenho, desde a configuração dos testes, à obtenção e interpretação dos resultados.

4.3.1 Teste de Escalabilidade da Plataforma

O próximo cenário de testes da plataforma passa pela execução de um teste de escalabilidade, ou seja, um teste para verificar como é que o sistema reage à instanciação de múltiplos *quadcopters* em simultâneo no mesmo ambiente virtual.

Este cenário vai ser composto por um cenário controlado e sem nenhum objeto para além dos *quadcopters* por formaa não se influenciarem os resultados. Vão sendo instanciados *quadcopters* com intervalos curtos de tempo de forma a que a plataforma estabilize os *frames* por segundo.

O objetivo será verificar o nível de processamento através das FPS que consegue processar com o número de *quadcopters* instanciados. Neste teste foi apenas considerada a instanciação dos *quadcopters* sem estarem a executar missões, permitindo ver os consumos mínimos que estes apresentam, foram também analisados na situação com o motor de física ligado e posteriormente desligado.

Os resultados obtidos encontram-se apresentados nos dois gráficos da Figura 4.5. No lado esquerdo apresentam-se os valores recolhidos (de uma pequena amostra) com o motor da física ligado nas duas configurações de acordo com as legendas e conforme pode ser observado, foi aplicado sobre os pontos de cada série de dados uma linha de tendência com uma curva exponencial

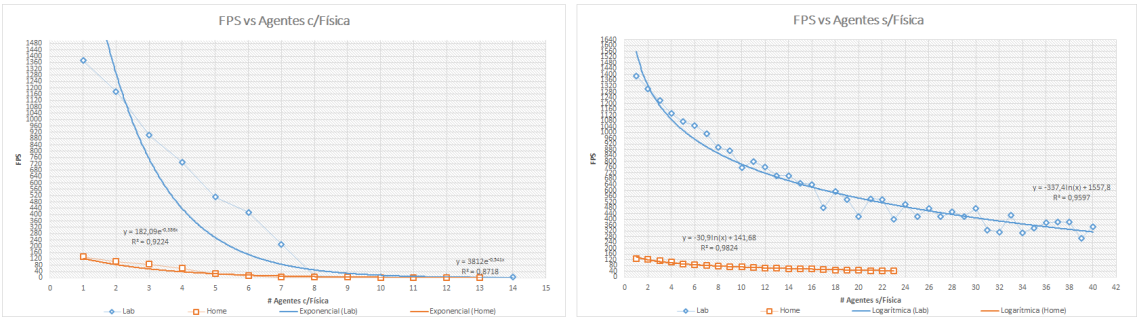


Figura 4.5: Resultados: FPS vs Agentes

à qual se ajustavam melhor os resultados. Através da equação 4.1 calculada pode-se verificar que qualquer valor superior a 13 agentes as FPS da plataforma vão estar próximas de zero na configuração "Home", para a equação 4.2 pode-se verificar que qualquer valor superior a 31 as FPS vão estar também próximas de zero.

$$182.09 * e^{-0.386*x} = FPS(Home) \tag{4.1}$$

$$3812 * e^{-0.341*x} = FPS(Lab) \tag{4.2}$$

No caso da plataforma com o motor de física desligado representado pelo gráfico do lado direito da figura, foi aplicado sobre os pontos uma linha de tendência logarítmica que permitiu um melhor ajuste do que a exponencial. Através da equação 4.3 calculada pode-se verificar que qualquer valor superior a 98 as FPS da plataforma vão estar próximas de zero, no caso da equação 4.4, o número de agentes instanciados vai ser bastante superior.

$$-30.9 * ln(x) + 141.68 = FPS(Home) \tag{4.3}$$

$$-337.4 * ln(x) + 1557.8 = FPS(Lab) \tag{4.4}$$

Através dos resultados obtidos permitiu verificar-se na plataforma de simulação, que quantas mais instâncias estiverem a ser executadas em paralelo pior se torna o tempo de resposta do ambiente virtual com o motor de física em execução. Isto deve-se em grande parte à integração da física nos *quadcopters* que necessitam de mais processamento, para cálculos da gravidade, acelerações, velocidades, etc. O nível de precisão definido para o motor de física e as deteções de colisões também fazem diminuir as FPS, isto verificou-se comparando com os dados do gráfico do lado direito. Caso para a simulação só se pretenda testar cenários de cooperação e estratégias entre agentes é preferível desligar a física para se conseguir testar as estratégias com grupos maiores de *quadcopters* virtuais. Para se desligar a física do simulador deve-se trocar a configuração na interface do utilizador.

Também se verificou que quanto melhor for a configuração do computador, melhor será o desempenho da plataforma como se conseguiu observar nos resultados das duas configurações apresentadas na Tabela 4.1, pelo que a utilização da plataforma com agentes distribuídos também iria otimizar ainda mais a simulação.

4.3.2 Teste de Desempenho da Plataforma

Por último apresenta-se o desempenho da plataforma no que diz respeito ao nível da memória, das *threads* e dos *hot spots* que apresentam. O desempenho obtido em tempo de execução foi capturado através da criação de um *profile* da plataforma.

Em relação à memória (*memory heap*), foi analisado a taxa de utilização de memória desde o arranque da plataforma até à instanciação de cinco *quadcopters*. Na Figura 4.6 verifica-se que

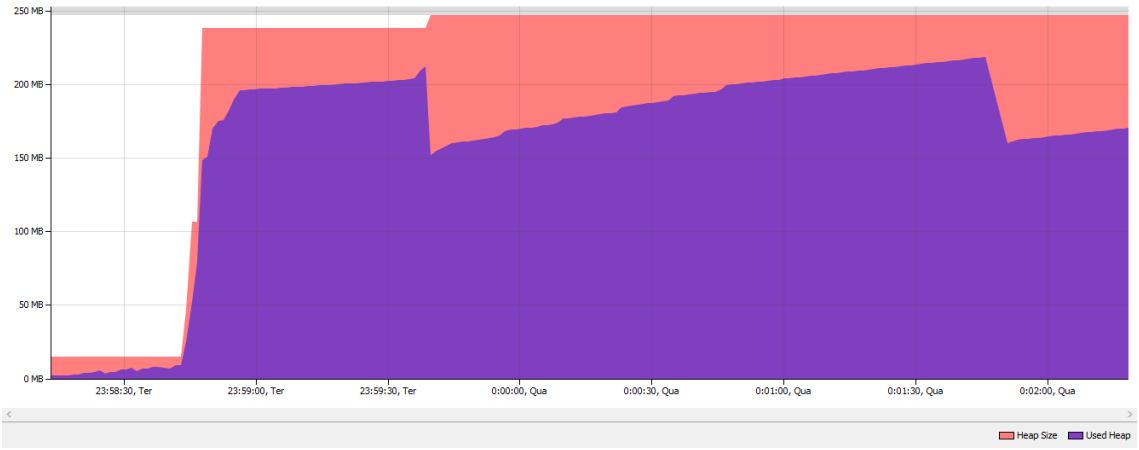


Figura 4.6: Desempenho da Plataforma: *Memory Heap*

entre o instante 23:58:45 e o instante 23:58:55 existiu um consumo substancial de utilização de memória, devido ao arranque de todos os componentes da plataforma desde a interface ao ambiente virtual. De seguida o processamento estabilizou e removeu os recursos que já não são necessários do carregamento de todo o sistema como se observa no pico em que a memória desceu até as 150MB.

Após esta estabilização foram adicionados cinco *quadcopters* em curtos intervalos de tempo para a estabilização de cada *quadcopter*. Entre o instante 23:59:45 e o instante 00:00:50 foram então instanciados os vários *quadcopters* gastando em média cada um 9,5MB de memória.

De novo perto do fim do gráfico houve uma nova quebra no uso da memória, isto deve-se ao facto de quando a plataforma não estar a ser visualizada ou quando perde o foco e esta não processa a renderização da imagem para o ecrã de forma a optimizar os recursos do sistema.

O volume de *threads* criadas e de classes carregadas está relacionado também com a memória utilizada, uma vez que para cada *thread* é alocado todos os recursos associados a um *quadcopter*

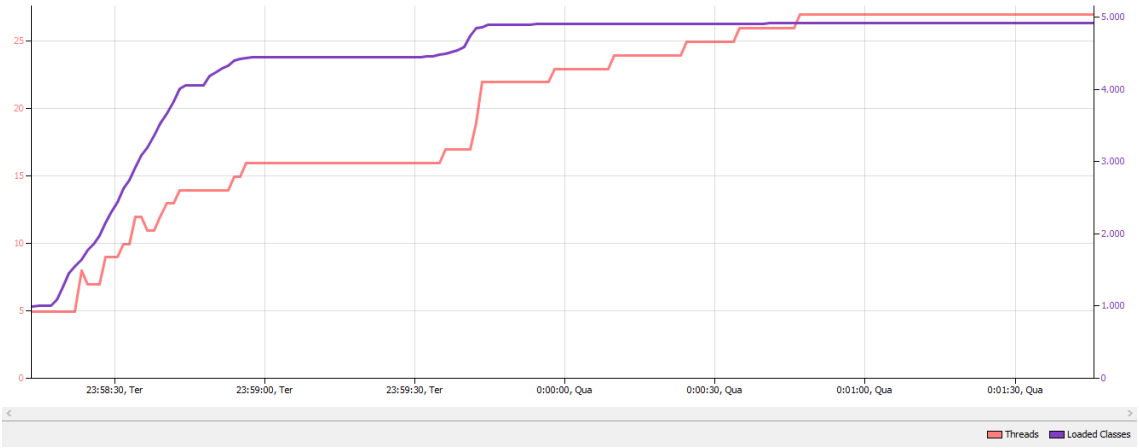


Figura 4.7: Desempenho da Plataforma: *Threads/Classes*

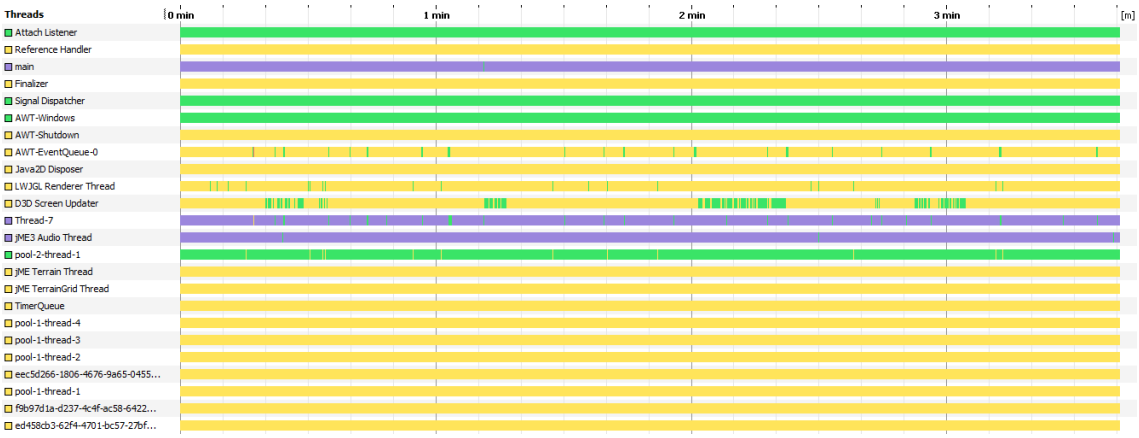


Figura 4.8: Desempenho da Plataforma: *Threads*

virtual. Na Figura 4.7 pode-se constatar que nos mesmos tempos de instanciação dos *quadcopters* apresentado no gráfico anterior, também foi criada uma *thread* para cada *quadcopter* e as classes necessárias para a sua instanciação são as mesmas, visto que foram carregadas inicialmente e não se verificou o seu aumento à medida que foram instanciados, isto porque os *quadcopters* instanciados eram todos do mesmo tipo. Na execução da plataforma foram carregadas em memória cerca de cinco mil classes e vinte e sete *threads*, cinco das quais para os agentes.

Na Figura 4.8 estão apresentadas as threads do sistema onde se consegue ter uma melhor perceção da sua execução. Como se pode observar, a maior parte do tempo as *threads* estão em modo de espera (cor amarela), situação em que se encontram os *quadcopters* que foram instanciados e se encontravam parados na sua posição. Algumas destas threads são as responsáveis pela atualização da interface pelo que vão fazendo atualizações regulares demonstrando pelas riscas verdes nas barras amarelas. As threads que se encontram em modo de execução (cor verde) são as responsáveis pela apresentação da GUI, captura de eventos e a *pool-2-thread-1* está responsável por toda a sincronização do ambiente virtual.

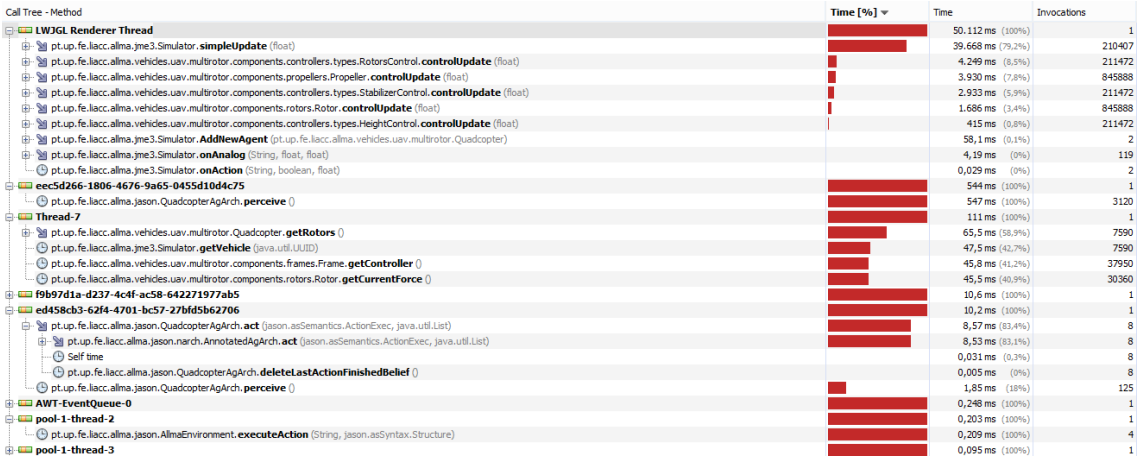


Figura 4.9: Desempenho da Plataforma: *Hot Spots*

Por último verificaram-se os *hot spots* da plataforma. Na Figura 4.9 verifica-se que o sistema tem um hot spot associado ao *simpleUpdate()*, este método permite atualizar o estado interno do *quadcopter* a partir das informações obtidos pelo modelo 3D, pelo que é natural que consuma maior processamento.

Os restantes pontos críticos encontram-se em cada *thread* que representa um agente, através dos métodos *perceive()* ou *executeAction()*, estes são os métodos que são executados pelo motor cognitivo do Jason. Sendo também esperado este processamento mais exaustivo por parte dos agentes, uma vez que estão em constante atualização.

4.4 Conclusão

Neste capítulo foram descritos cinco cenários de teste que ilustram o funcionamento e desempenho da plataforma. A partir deste pequeno conjunto de testes efetuados à plataforma de simulação permitiu-se verificar os pontos mais fracos a serem melhorados futuramente.

Na simulação simbiótica, em relação à ligação entre os dois sistemas e através dos módulos XBee. Quando existia falha na troca dos pacotes era observável no simulador o salto entre as posições das imagens, isto é, se o teste executado num ambiente controlado apresentava estes problemas, então num teste em ambiente não controlado seria ainda mais crítico o problema. Pois no ambiente não controlado existem muitos outros fatores que podem influenciar a comunicação.

Em relação ao teste de simulação dos sensores de infravermelhos conseguiu-se observar que apesar do sensor real perder a sua eficácia devido ao aumento da distância dos objetos, esta poderia ser compensada no simulador. Através do uso dos dados simulados, se for conhecida quer a posição do *quadcopter* virtual, quer do real pode-se indicar ao simulador para utilizar as distâncias obtidas no ambiente virtual.

Por último, foi analisado o desempenho da plataforma em relação a pontos críticos, relativos à própria utilização e libertação de memória e à execução de *threads* uma vez que em SMA é normal se tirar partido do processamento em paralelo.

Nomeadamente ao nível da otimização do processamento, conseguiu-se verificar que seria uma boa solução fazer a distribuição dos agentes em outras máquinas, deixando o simulador numa máquina independente, só com o propósito de representar os agentes e permitir controlá-los remotamente.

Capítulo 5

Conclusão

Neste capítulo são apresentadas as considerações finais, as satisfações dos objetivos, os possíveis desenvolvimentos futuros e as lições retiradas a partir da reflexão sobre todo o trabalho desenvolvido.

5.1 Considerações Finais

Após o estudo dos fundamentos teóricos e do estado da arte, verificou-se o funcionamento e as capacidades que este tipo de plataforma deveria ter e a forma como tudo deve interagir. Dessa forma, esse estudo permitiu criar uma plataforma bem estruturada e capaz de interligar sistemas distintos como simulação de entidades virtuais, agentes e *quadcopters* reais.

O balanço da proposta de dissertação é bastante positivo uma vez que na sua totalidade se conseguiu cumprir todos os objetivos a desenvolver na dissertação. Por este trabalho ser muito extenso e permitir implementar muitas funcionalidades houve a necessidade de se identificarem os pressupostos. Desta forma, conseguem-se definir com mais rigor os limites ao nível de detalhes do trabalho que realmente deve ser desenvolvido numa fase inicial e o que mais tarde se pode melhorar em desenvolvimentos futuros.

Durante o decorrer da dissertação também foi necessário um trabalho contínuo e planeado para que os resultados fossem os apresentados. No âmbito do desenvolvimento da plataforma, conseguiu-se identificar quais as tecnologias, os sistemas e algoritmos a utilizar, assim como se reúnem todos os componentes.

Embora se tenha obtido sucesso na conceção e implementação da plataforma, existe ainda um extenso trabalho a realizar para tornar esta plataforma mais versátil e mais completa em termos de funcionalidades. A Tabela 5.1 apresenta uma análise das Potencialidades, Fraquezas, Oportunidades e Ameaças (PFOA) do projeto.

Tabela 5.1: Análise PFOA

	Útil (Para atingir o objetivo)	Prejudicial (Para atingir o objetivo)
Origem Interna (Atributos do Projeto)	**Forças** • Vantagem competitiva em relação a plataformas existentes; • Ligação simbiótica entre o *quadcopter* real/virtual; • *Quadcopters* autónomos; • Integração de simulação de múltiplos VANTs;	**Fraquezas** • Não ter um bom *background*; • As tecnologias não corresponderem as necessidades esperadas; • Parte do trabalho estar dependente de terceiros;
Origem Externa (Atributos do Ambiente)	**Oportunidades** • Plataforma mais robusta de simulação; • Permite uma melhor otimização das tomadas de decisão; • Base para o desenvolvimento de muitas outras funcionalidades futuras;	**Riscos** • Expectativas muito elevadas; • Desenvolvimento mais demorado que o previsto; • Problemas encontrados durante o desenvolvimento;

5.2 Melhoramentos

O estado atual da plataforma de simulação permite configurar e instanciar um *quadcopter* num ambiente importado através de um modelo ou utilizar o genérico de exemplo disponibilizado para testes. O *quadcopter* pode ser controlado a partir dos agentes, do controlo manual e ainda tem a hipótese de selecionar o avatar de um equipamento real.

Devido ao facto de ser uma plataforma de simulação desenvolvida de raiz, ainda existem lacunas e melhoramentos a serem feitos, em especial na dificuldade de controlo do *quadcopter* com os controladores PID. Para se contornar este problema, numa fase inicial foi utilizada uma alternativa para executar os movimentos do VANT uma vez que é necessário ter muita experiência para se conseguir implementar este tipo de controlo que não foi tido em conta no planeamento.

Os sensores também foram implementados de uma forma simples, apesar de considerarem algum erro, típicos dos sensores eletrônicos ainda podem ser mais realistas. Caso se deseje na simulação que os sensores simulem o comportamento dos sensores reais, existe a possibilidade de se adicionar a equação da curva característica apresentada nas fichas técnicas dos componentes durante a instanciação dos sensores nos *quadcopters* virtuais.

Apesar da componente dos SMA já estar integrada, estes ainda podem ser melhorados aumentando a quantidade das informações disponibilizadas sobre os *quadcopters* nas perceções dos agentes. Quanto mais informações reunirem e melhores planos se implementar, melhor serão as respostas do sistema incluindo as respostas dos próprios agentes. Apesar de ainda ser uma plataforma em fase inicial, poderá a vir a ser uma boa ferramenta de investigação para o desenho, implementação e testes de agentes VANTs utilizados no domínio das aplicações civis.

5.3 Trabalho Futuro

Vários podem ser os desenvolvimentos a serem futuramente integrados nesta plataforma, uma vez que se encontra implementada de forma modular. Começando pelo próprio desenvolvimento de mecanismos para a resolução de tarefas complexas de coordenação e cooperação. Estas podem ser implementadas através dos planos que cada agente vai interpretar. De seguida poderia-se melhorar ainda mais o modelo do *quadcopter* já implementado, acrescentando ainda mais parâmetros e otimizando as fórmulas utilizadas nos cálculos que obtivessem resultados ainda mais precisos. Uma vez feitos estes melhoramentos, poder-se-ia proceder à implementação de outros veículos autónomos na plataforma de simulação, dado que só seria necessário reutilizar o código existente e criar o modelo gráfico e analítico para o novo veículo.

Implementando todas estas melhorias previamente referidas, seria então necessário desenvolver e implementar uma ontologia própria e um protocolo de comunicação onde os vários tipos de veículos conseguissem perceber o significado de cada instrução. Fazendo um mapeamento direto entre as características dos vários veículos e também dos termos utilizados em cada uma das áreas de domínio, por exemplo os VANTs, VTNTs, *etc*.

Muitas outras melhorias poderiam ter sido implementadas, pelo que foram referidas as que serão mais benéficas nesta fase inicial da plataforma de simulação. Por último, seria também interessante fazer otimizações ao nível do processamento dos dados, ou seja, uma vez que um simulador integra vários agentes num mesmo local e conjugando todos os dados que tem que ser processados em simultâneo, é obvio que o processador vai perder eficácia com tantas *threads*.

Apesar de estar um pouco fora do âmbito desta dissertação, seria uma ótima otimização separar todas as *threads* e processamento dos controladores utilizando o processamento dos núcleos das placas gráficas mais recentes que tenham a arquitetura *Compute Unified Device Architecture* (CUDA) disponível. Existem já várias ferramentas que ajudam na sua implementação, por exemplo o JCuda é uma plataforma Java para a interação com o CUDA. Assim através da aplicação de um estudo e dos respetivos testes poder-se-ia aumentar a escalabilidade e a robustez da própria plataforma tirando partido dos recursos disponíveis do sistema de computação.

5.4 Lições Aprendidas

Embora se tenha concluído com sucesso a integração de todas as tecnologias na plataforma de simulação, existe ainda um extenso trabalho a realizar ao nível das otimizações e correções de problemas que normalmente surgem em sistemas criados de raiz. Normalmente estes problemas devem-se em grande parte à dificuldade de se definir e planearem todos os passos do desenvolvimento. Uma lição que se pode retirar é que por mais planeamento que se faça e conhecimento que se possua sobre as ferramentas que se vão utilizar, surgem sempre pequenos detalhes que vão atrasar ou impossibilitar o desenvolvimento da plataforma da forma desejada.

Contudo, o presente protótipo da plataforma de simulação dado ter adotado uma abordagem modular devido às ferramentas e tecnologias selecionadas, vai permitir o seu desenvolvimento no âmbito da integração e manutenção de novas funcionalidades e sensores de uma forma simplificada, criando desta forma uma plataforma de desenho, simulação e desenvolvimento bastante versátil.

Outra lição aprendida foi que se deve tentar aprofundar o planeamento um nível abaixo das funcionalidades esperadas do sistema, isto vai permitir ter uma melhor perceção do tempo que se vai demorar em cada tarefa e ao mesmo tempo consegue-se explorar as capacidades e o funcionamento da tecnologia. Pois, mais vale perder um pouco mais de tempo durante o planeamento do que ter que se alterar no decorrer do projeto. Por último também foi verificado que não se devem planear tarefas dependentes de outros fatores, ou tomar por garantido que tudo vai funcionar ou estar pronto na altura certa. Na pior das situações, deve-se ter sempre elaborado um plano de contingência para estas situações.

Referências

[AMM06] Christopher J Augeri, Kevin M Morris e Barry E Mullins. Harvest: a framework and co-simulation environment for analyzing unmanned aerial vehicle swarms. In *Military Communications Conference, 2006. MILCOM 2006. IEEE*, pages 1–7. IEEE, 2006.

[ATCL08] Heiko Aydt, Stephen John Turner, Wentong Cai e Malcolm Yoke-Hean Low. Symbiotic simulation systems: An extended definition motivated by symbiosis in biology. In *Principles of Advanced and Distributed Simulation, 2008. PADS'08. 22nd Workshop on*, pages 109–116. IEEE, 2008.

[ATCL09a] H. Aydt, S.J. Turner, Wentong Cai e M.Y.H. Low. Research issues in symbiotic simulation. In *Simulation Conference (WSC), Proceedings of the 2009 Winter*, pages 1213–1222, 2009. doi:10.1109/WSC.2009.5429419.

[ATCL09b] Heiko Aydt, Stephen John Turner, Wentong Cai e Malcolm Yoke Hean Low. Research issues in symbiotic simulation. *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 1213–1222, December 2009. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5429419`, doi:10.1109/WSC.2009.5429419.

[BH06] Rafael H Bordini e Jomi F Hübner. Bdi agent programming in agentspeak using jason. In *Computational logic in multi-agent systems*, pages 143–164. Springer Berlin Heidelberg, 2006.

[BNS04] S Bouabdellah, A Noth e R Siegwart. modelling of the (os4) quadrotor" modelling course. *EPFL, may*, 2004.

[BPMR08] Rodrigo AM Braga, Marcelo Petry, Antonio Paulo Moreira e Luis Paulo Reis. A development platform for intelligent wheelchairs for disabled people. In *International Conference on Informatics in Control, Automation and Robotics*, pages 115–121, 2008.

[Bro91] Rodney Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[Bru91] Jose C. Brustoloni. Autonomous agents: Characterization and requirements. Technical report CMUCS-91-204, School of Computer Science, Carnegie Mellon University, November 1991.

[CC05] Shyamal Suhana Chandra e Kailash Chandra. A comparison of java and c#. *J. Comput. Sci. Coll.*, 20(3):238–254, February 2005. URL: `http://dl.acm.org/citation.cfm?id=1040196.1040228`.

[CH03] Ming Chen e Mihai Huzmezan. A combined mbpc/2 dof h ∞ controller for a quad rotor uav. In *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, 2003.

[CL04] Joshua J Corner e Gary B Lamont. Parallel simulation of uav swarm scenarios. In *Proceedings of the 36th conference on Winter simulation*, pages 355–363. Winter Simulation Conference, 2004.

[CLW+07] Stefano Carpin, Michael Lewis, Jijun Wang, Stephen Balakirsky e Chris Scrapper. Usarsim: a robot simulator for research and education. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1400–1405. IEEE, 2007.

[CNF09] G. Campa, M.R. Napolitano e M.L. Fravolini. Simulation environment for machine vision based aerial refueling for uavs. *Aerospace and Electronic Systems, IEEE Transactions on*, 45(1):138–151, 2009. doi:10.1109/TAES.2009.4805269.

[Dem91] Y. Demazeau. Coordination patterns in multi-agent worlds applications to computer vision and robotics. In *Intelligent Agents, IEE Colloquium on*, pages 5/1 –5/9, feb 1991.

[DPBW+08] H. Dyke Parunak, Sven Brueckner, Danny Weyns, Tom Holvoet, Paul Verstraete e Paul Valckenaers. E pluribus unum: Polyagent and delegate mas architectures. In Luis Antunes, Mario Paolucci e Emma Norling, editors, *Multi-Agent-Based Simulation VIII*, volume 5003 of *Lecture Notes in Computer Science*, pages 36–51. Springer Berlin Heidelberg, 2008.

[DR94] Edmund H Durfee e Jeffrey S Rosenschein. Distributed problem solving and multi-agent systems: Comparisons and examples. *Ann Arbor*, 1001:48109, 1994.

[DW03] Ian Dickinson e Michael Wooldridge. Towards practical reasoning agents for the semantic web. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 827–834, New York, NY, USA, 2003. ACM. URL: `http://doi.acm.org/10.1145/860575.860708`, doi:10.1145/860575.860708.

[Ead01] Marc Eaddy. C# versus java. *Dr. Dobb's Journal*, page 74ff, 2001.

[FG97] Stan Franklin e Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, ECAI '96, pages 21–35, London, UK, UK, 1997. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=648203.749270`.

[FGI09] P.G. Fahlstrom, T.J. Gleason e UAV Systems Incorporated. *Introduction to UAV Systems*. UAV Systems, Incorporated, 2009. URL: `http://books.google.pt/books?id=RDjPQwAACAAJ`.

[Flo98] Adina Magda Florea. Introduction to multi-agent systems. *International Summer School on Multi-Agent Systems, Bucharest*, 1998.

[GAÖ01] Nasser Ghasem-Aghaee e Tuncer I Ören. Simulation and agents: Exploring the synergy. 2001.

[Gas87] Les Gasser. Distribution and coordination of tasks among intelligent agents. In *First Scandinavian Conference on Artificial Intelligence*, Tromsø, Normay, March 1987.

[GK02] Les Gasser e Kelvin Kakugawa. Mace3j: fast flexible distributed simulation of large, large-grain multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pages 745–752. ACM, 2002.

[GSGA09] Rahul Goel, Sapan M Shah, Nitin K Gupta e N Ananthkrishnan. Modeling, simulation and flight testing of an autonomous quadrotor. In *IISc Centenary International Conference and Exhibition on Aerospace Engineering, ICEAE, Bangalore, India*, pages 18–22, 2009.

[GSRO08] Ricardo Gimenes, Daniel Castro Silva, Luís Paulo Reis e Eugénio Oliveira. Flight simulation environments applied to agent-based autonomous uavs. In *ICEIS (4)*, pages 243–246, 2008.

[HBA11] Marcelo T Hama, Rafael H Bordini e Rodrigo S Allgayer. Agentspeak for uav development: Towards a technology bridge. 2011.

[HI91] Carl Hewitt e Jeff Inman. Dai betwixt and between: fromintelligent agents' to open systems science. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(6):1409–1419, 1991.

[HRHL01] Nick Howden, Ralph Rönnquist, Andrew Hodgson e Andrew Lucas. Jack intelligent agents-summary of an agent infrastructure. In *5th International conference on autonomous agents*, 2001.

[HRW+04] G. Hoffmann, D.G. Rajnarayan, S.L. Waslander, D. Dostal, Jung Soon Jang e C.J. Tomlin. The stanford testbed of autonomous rotorcraft for multi agent control (starmac). In *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, volume 2, pages 12.E.4–121–10 Vol.2, 2004. doi:10.1109/DASC.2004.1390847.

[Hub99] Marcus J. Huber. Jam: A bdi-theoretic mobile agent architecture. In *Proceedings of the Third Annual Conference on Autonomous Agents*, AGENTS '99, pages 236–243, New York, NY, USA, 1999. ACM. URL: `http://doi.acm.org/10.1145/301136.301202`, doi:10.1145/301136.301202.

[Jac99] Ivar Jacobson. *The unified software development process*. Pearson Education India, 1999.

[Jac07] J. Jackson. Microsoft robotics studio: A technical introduction. *Robotics Automation Magazine, IEEE*, 14(4):82–87, 2007. doi:10.1109/M-RA.2007.905745.

[JCWW10] Kai-Yuan Jheng, Chih-Hao Chao, Hao-Yu Wang e An-Yeu Wu. Traffic-thermal mutual-coupling co-simulation platform for three-dimensional network-on-chip. In *VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on*, pages 135–138. IEEE, 2010.

[Jen96] N.R. Jennings. Applying agent technology. Plenary presentation at PAAM'96, 1996.

[JF01] Eric N Johnson e Sébastien Fontaine. Use of flight simulation to complement flight testing of low-cost uavs. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, pages 6–9, 2001.

[JKRM12] João Pedro Jorge, Zafeiris Kokkinogenis, Rosaldo JF Rossetti e Manuel AP Marques. Simulation of an order picking system in a pharmaceutical warehouse. In *SIMUL 2012, The Fourth International Conference on Advances in System Simulation*, pages 107–112, 2012.

[JPC+12] M. Jakob, M. Pechoucek, M. Cap, O. Vanek e P. Novak. Towards incremental development of human-agent-robot applications using mixed-reality testbeds. *Intelligent Systems, IEEE*, PP(99):1–1, 2012. doi:10.1109/MIS.2012.7.

[JSW98] Nicholas R. Jennings, Katia Sycara e Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, January 1998. URL: `http://dx.doi.org/10.1023/A:1010090405266`, doi:10.1023/A:1010090405266.

[KA07] Farzad Kamrani e Rassul Ayani. Using On-line Simulation for Adaptive Path Planning of UAVs. *11th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'07)*, pages 167–174, October 2007. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4384545`, doi:10.1109/DS-RT.2007.23.

[KD07] JE Kuchar e Ann C Drumm. The traffic alert and collision avoidance system. *Lincoln Laboratory Journal*, 16(2):277, 2007.

[KH04] N. Koenig e A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154 vol.3, 2004. doi:10.1109/IROS.2004.1389727.

[Kiv06] Arda Özgür Kivrak. Design of control systems for a quadrotor flight vehicle equipped with inertial sensors. *Atilim University, December*, 2006.

[LC88] V.R. Lesser e D.D Corkill. The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks. *Readings from the AI Magazine*, 1-5:69–85, 1988. URL: `http://mas.cs.umass.edu/paper/339`.

[Lee08] E.A. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369, 2008. doi:10.1109/ISORC.2008.25.

[LLL+05] Malcolm Yoke Hean Low, Kong Wei Lye, Peter Lendermann, Stephen John Turner, Reman Tat Wee Chim e Surya Hadisaputra Leo. An agent-based approach for managing symbiotic simulation of semiconductor assembly and test operation. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 85–92. ACM, 2005.

[MCI+09] Adriano Mancini, Andrea Cesetti, A Iuale, Emanuele Frontoni, Primo Zingaretti e Sauro Longhi. A framework for simulation and testing of uavs in cooperative scenarios. In *Unmanned Aircraft Systems*, pages 307–329. Springer, 2009.

[MDA05] Viviana Mascardi, Daniela Demergasso e Davide Ancona. Languages for programming bdi-style agents: an overview. In *WOA*, pages 9–15. Citeseer, 2005.

[MF09] Drogoul A. Michel F., Ferber J. *Multi-Agent Systems: Simulation and Applications*. CRC Press, adelinde m. uhrmacher, university of rostock, germany; danny weyns, katholieke universiteit leuven, belgium edition, 2009.

[Mic04] Olivier Michel. Webotstm: Professional mobile robot simulation. *arXiv preprint cs/0412052*, 2004.

[MKSP13] Jose Macedo, Zafeiris Kokkinogenis, Guilherme Soares e Deborah Perrotta. A hla-based multi-resolution approach to simulating electric vehicles in simulink and sumo. In *16th International IEEE Annual Conference on Intelligent Transportation Systems, The Hague, The Netherlands*, 2013.

[oT11] U.S. Department of Transportation. *Introduction to TCAS II Version 7.1*. HQ111358. Federal Aviation Administration, 2 2011.

[OVDPFB03] James J. Odell, H. Van Dyke Parunak, Mitch Fleischer e Sven Brueckner. Modeling agents and their environment. In *Proceedings of the 3rd international conference on Agent-oriented software engineering III*, AOSE'02, pages 16–31, Berlin, Heidelberg, 2003. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1754726.1754729`.

[PBL05] Alexander Pokahr, Lars Braubach e Winfried Lamersdorf. Jadex: A bdi reasoning engine. In *Multi-agent programming*, pages 149–174. Springer, 2005.

[PMRR11] Marcelo Petry, Antonio Paulo Moreira, Luis Paulo Reis e Rosaldo Rossetti. Intelligent wheelchair simulation: Requirements and architectural issues. In *11th International Conference on Mobile Robotics and Competitions, Lisbon*, 2011.

[PR12] José LF Pereira e Rosaldo JF Rossetti. An integrated architecture for autonomous vehicles simulation. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 286–292. ACM, 2012.

[PRG11] Lúcio S Passos, Rosaldo JF Rossetti e Joaquim Gabriel. An agent methodology for processes, the environment, and services. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 2124–2129. IEEE, 2011.

[PRK11] Lucio Sanchez Passos, Rosaldo JF Rossetti e Z Kokkinogenis. Towards the next-generation traffic simulation tools: a first appraisal. In *Information Systems and Technologies (CISTI), 2011 6th Iberian Conference on*, pages 2367–2372. IEEE, 2011.

[Ör02] Tuncer Ören. *Future of Modelling and Simulation: Some Development Areas*. Summer Computer Simulation Conference, 2002.

[Ör10] Tuncer Ören. *Modeling and Simulation: A Comprehensive and Integrative View*, pages 1–36. Wiley-VCH Verlag GmbH & Co. KGaA, 2010. URL: http://dx.doi.org/10.1002/9783527627783.ch1, doi:10.1002/9783527627783.ch1.

[RG10] Syed Ali Raza e Wail Gueaieb. Intelligent flight control of an autonomous quadrotor. *InTech*, 2010.

[RHB07] Michael Wooldridge Rafael H. Bordini, Jomi Fred Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, Ltd, 2007.

[RN03] Stuart J. Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[ROB07] Rosaldo JF Rossetti, Eugénio C Oliveira e Ana LC Bazzan. Towards a specification of a framework for sustainable transportation analysis. *13th Portuguese Conference on Artificial Intelligence, Guimarães, Portugal*, 2007.

[RPV11] Alessandro Ricci, Michele Piunti e Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.

[RYAS09] Osamah A Rawashdeh, Hong Chul Yang, Rami D AbouSleiman e Belal H Sababha. Microraptor: A low-cost autonomous quadrotor system. In *Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, California, USA*, 2009.

[SA01] Zak Sarris e STN ATLAS. Survey of uav applications in civil markets (june 2001). In *The 9 th IEEE Mediterranean Conference on Control and Automation (MED'01)*, 2001.

[Sán97] J Alfredo Sánchez. A taxonomy of agents. *Rapport technique, ICT-Universidad de las Américas-Puebla, México*, 1997.

[SMD12] Angela P. Schoellig, Fabian L. Mueller e Raffaello D'Andrea. Optimization-based iterative learning for precise quadrocopter trajectory tracking. *Autonomous Robots*, 33(1-2):103–127, April 2012. URL: http://link.springer.com/10.1007/s10514-012-9283-2, doi:10.1007/s10514-012-9283-2.

[Smi80] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, C-29(12):1104–1113, 1980. doi:10.1109/TC.1980.1675516.

[SV00] Peter Stone e Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000. URL: http://dx.doi.org/10.1023/A%3A1008942012299, doi:10.1023/A:1008942012299.

[TGP+08] Ying Tan, Steve Goddard, Lance C Perez et al. A prototype architecture for cyber-physical systems. *SIGBED Review*, 5(1):26, 2008.

[TLFA09] Beau J Tippetts, Dah-Jye Lee, Spencer G Fowers e James K Archibald. Real-time vision sensor for an autonomous hovering micro unmanned aerial vehicle. *Journal of Aerospace Computing, Information, and Communication*, 6(10):570–584, 2009.

[TSLGDCPB13] Jorge Torres-Sánchez, Francisca López-Granados, Ana Isabel De Castro e José Manuel Peña-Barragán. Configuration and specifications of an unmanned aerial vehicle (uav) for early site specific weed management. *PLoS ONE*, 8(3):e58210, 03 2013. URL: http://dx.doi.org/10.1371%2Fjournal.pone.0058210, doi:10.1371/journal.pone.0058210.

[TU10] Andreas Tolk e Adelinde M. Uhrmacher. *Agents: Agenthood, Agent Architectures, and Agent Taxonomies*, pages 73–109. Wiley-VCH Verlag GmbH & Co. KGaA, 2010. URL: http://dx.doi.org/10.1002/9783527627783.ch3, doi:10.1002/9783527627783.ch3.

[UW09] Adelinde M. Uhrmacher e Danny Weyns. *Multi-Agent Systems: Simulation and Applications*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2009.

[Vla07] Nikos Vlassis. A concise introduction to multiagent systems and distributed. In *Artificial Intelligence. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers*, 2007.

[WA06] Kong Wai Weng e MSB Abidin. Design and control of a quad-rotor flying robot for aerial surveillance. In *Research and Development, 2006. SCOReD 2006. 4th Student Conference on*, pages 173–177. IEEE, 2006.

[WBKS02] Joachim Wahle, Ana Lúcia C Bazzan, Franziska Klügl e Michael Schreckenberg. The impact of real-time information in a two-route scenario using agent-based simulation. *Transportation Research Part C: Emerging Technologies*, 10(5–6):399 – 417, 2002. URL: http://www.sciencedirect.com/science/article/pii/S0968090X02000311, doi:http://dx.doi.org/10.1016/S0968-090X(02)00031-1.

[WOO07] Danny Weyns, Andrea Omicini e James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14:5–30, 2007. URL: http://dx.doi.org/10.1007/s10458-006-0012-0.

[WSR+05] DANNY WEYNS, MICHAEL SCHUMACHER, ALESSANDRO RICCI, MIRKO VIROLI e TOM HOLVOET. Environments in multiagent systems. *The Knowledge Engineering Review*, 20:127–141, 5 2005. URL: http://journals.cambridge.org/article_S0269888905000457, doi:10.1017/S0269888905000457.

[ZLWZ11] Hongtao Zhang, Longqiu Li, Lin Wang e Guangyu Zhang. Vision-assisted hovering control for an indoor micro aerial robot. In *Electronics, Communications and Control (ICECC), 2011 International Conference on*, pages 581–584. IEEE, 2011.

REFERÊNCIAS

Apêndice A

Ambiente de Desenvolvimento

Neste apêndice descreve-se a instalação do ambiente de desenvolvimento, desde a análise das tecnologias disponíveis à seleção da linguagem de programação e ferramentas de desenvolvimento utilizadas.

A.1 Seleção das Ferramentas de Desenvolvimento

Uma plataforma de simulação de robótica normalmente é projetada para integrar aplicações para veículos específicos, assim como VANTs, sem estar dependente do veículo real. Em alguns casos, estas aplicações podem ser transferidas para o veículo real com pequenas alterações ou adaptações por forma a serem integradas no equipamento real. Os simuladores permitem assim reproduzir situações que simplesmente não podem ser criadas no mundo real devido a vários fatores, como custos, tempo e recursos limitados.

Um simulador também deve permitir a prototipagem rápida, uma vez que a sua aplicação passe a desenvolver um sistema capaz de duplicar processos para um ambiente específico em uma variedade de situações, incluindo também a própria dinâmica do veículo. Esta nova plataforma de simulação em desenvolvimento pode agregar um determinado valor ao conseguir simular as características específicas de um VANT através do uso de sistemas de agentes autónomos, por forma a executar no mundo real as decisões tomadas pelo simulador. Permitindo também remover todo este processamento pesado à parte dos poucos recursos da placa controladora do VANT.

Para tal é necessário definir que ferramentas estão disponíveis para o desenvolvimento da plataforma de simulação. Começando pelas ferramentas do desenvolvimento do simulador, desde logo pode-se verificar que existem vários motores 3D disponíveis, capazes de implementar este projeto. Após algum estudo sobre as características de cada motor gráfico, foram identificados três potenciais candidatos que se destacaram entre os restantes, sem querer excluir outros programas como por exemplo o Gazebo.

Tabela A.1: Comparação de Motores Gráficos 3D

Motores Gráficos 3D	**Unity**	**jMonkeyEngine3 (jME3)**	**CryENGINE®3**
Nível do editor:	Complexo	Simples e intuitivo, integrado no Netbeans.	Sandbox integrado com FarCry.
Código Fonte:	Código fonte não incluído no SDK.	Código fonte incluído no SDK.	Código fonte não incluído no SDK.
SDK:	SDK inclui documentação.	SDK inclui documentação e vários módulos.	SDK inclui documentação e ferramentas adicionais.
Documentação:	SKD inclui guias e exemplos.	Guias, exemplos, e fóruns de comunidade ativos.	SKD inclui guias e exemplos.
Outros:	Necessária licença para acesso ao código fonte e documentação.	Código fonte aberto.	Permite a edição em tempo de execução. Necessária licença para acesso ao código fonte e documentação para aplicações comerciais.

Os fatores para a escolha da linguagem de programação e da ferramenta de desenvolvimento tiveram em atenção as necessidades de todo o projeto na globalidade, assim como a sua interoperabilidade entre todos os componentes e sistemas operativos. Para um rápido desenvolvimento foi definido à partida que se iria utilizar qualquer linguagem de programação orientada a objetos como C# e Java.

O Java tem vindo a ser uma das primeiras linguagens de programação utilizadas na maioria das universidades, apesar de se já começar a utilizar a linguagem C#. O Java inclui otimizações específicas da plataforma que a *Java Virtual Machine* (JVM) utiliza agora. A linguagem C# tem a mesma sintaxe geral, como C, C++ e Java. Além de que o C# é consideravelmente mais parecido com o Java que o próprio C++ [CC05].

Segundo alguns autores o C# é o Java com algumas características, inovações e com melhorias de eficácia. Para os programadores de Java, o C# compensa onde o Java perde alguma da sua eficácia fornecendo uma linguagem de alto-desempenho, orientada a objetos que se integra fortemente com o Windows. Entre Java e C#, as linguagens são honestamente muito similares apesar de em Java existirem certos comandos que não têm equivalentes no C# [Ead01].

Voltando aos três potenciais motores gráficos 3D, existem muitas características para se analisarem, na Tabela A.1 são apresentados os prós e contras sobre cada um dos motores escolhidos e de seguida são apresentados mais alguns detalhes sobre a implementação de cada um.

Alguns dos seguintes detalhes são baseados em opiniões da comunidade em relação ao Unity vs jME3. No Unity, a instanciação de novas entidades (GameObjects) tem que ser feito com um *script* "MonoBehaviour". Estes são os *scripts* que têm de ser associados a um GameObject

instanciado, não existe um "local de raiz" como void *main()* como se observa em desenvolvimento em C++ ou Java. Isto significa que a arquitetura de código ganha vantagem em termos de elegância e legibilidade e não existem assim enormes quantidades de fluxo de código lógico.

O licenciamento do Unity é gratuito mas, se o código fonte for necessário, é preciso pagar para se obter. Se for necessário todos os recursos do Unity Pro este torna-se muito dispendioso e em contrapartida o jME3 é totalmente grátis e de código fonte aberto.

O Unity não oferece soluções práticas para a proteção do código fonte. Com aplicações em Java, podemos usar ferramentas como *Excelsior JET*, *launch4j* ou *GCJ* para proteger o código compilando o código Java em um executável nativo para qualquer plataforma. Em termos de documentação, nem o Unity, nem jME são ideais e, quando existe falta de documentação, esta pode ser procurada nos fóruns de ajuda disponíveis. Ambos são bastante práticos e têm uma comunidade bastante ativa, disponibilizando respostas em tempo útil.

Em relação ao *multi-threading* para situações como IA e redes, o Unity não oferece nenhum acesso às *threads*. Quase todo o processamento acontece em uma única *thread* e está fora do controlo do programador. O jME3 nesta questão apenas não permite modificar nada que esteja relacionado com o *rendering* em *threads* externas.

Na perspetiva da implementação dos agentes autónomos, existem diversas plataformas que permitem a integração da arquitetura BDI. Na Tabela A.2 são apresentadas algumas vantagens e desvantagens de um pequeno conjunto de plataformas que foram estudadas, com base nas plataformas escolhidas na análise de Viviana Mascardi et al. [MDA05] no que diz respeito ao funcionamento e à comparação das várias plataformas todas elas implementadas em Java.

Três destas plataformas estudadas destacaram-se para a implementação dos agentes, uma é o *Java Agent DEvelopment Framework* (JADE) e outra é uma extensão chamada JADEX, por último o Jason também utilizado em vários projetos. Ambos usam Java e implementam a arquitetura de agentes com crenças, desejos e intenções (BDI). Isto permite desenvolver agentes capazes de executar planos sem decisões humanas.

A.2 Instalação do Ambiente de Desenvolvimento

A plataforma de desenvolvimento adotada foi um computador portátil com o sistema operativo Windows 8.1. Nesta plataforma instalou-se o ambiente de desenvolvimento padrão de Java, o ambiente de desenvolvimento de sistemas baseados em agentes e a plataforma de desenvolvimento do ambiente gráfico, assim como todas as ferramentas necessárias para a execução dos mesmos. Também se procedeu à instalação do programa de controlo da OpenPilot CC3D.

A.2.1 Ambiente de Desenvolvimento da Plataforma

Para o desenvolvimento da interface gráfica e respetivo ambiente virtual foi selecionado o jMonkeyEngine 3.0, uma extensão do IDE NetBeans e está focado para o desenvolvimento de interfaces baseada na linguagem de programação Java. É um programa de código-fonte livre e

Tabela A.2: Comparação de Plataformas BDI

Nome	Vantagens / Desvantagens
nuin	Nuin é uma arquitetura de agentes flexível e orientada para o desenvolvimento de agentes em aplicações web. Baseado numa arquitetura BDI foi concebida para dar o máximo de flexibilidade aos implementadores dos agentes enquanto mantém a integridade da sua arquitetura [DW03].
Jadex	Jadex é uma plataforma para a criação de agentes orientados por objetivos seguindo uma arquitetura BDI. Passam por disponibilizar um sistema simples com o objetivo de criar uma camada de agentes racional sobre a camada de *middleware* da infraestrutura do agente. Tentam dessa forma ultrapassar as limitações da arquitetura BDI introduzindo os objetivos explícitos e permitindo assim a agregação de novas funcionalidades ao nível da análise de resultados [PBL05].
JACK®	JACK também conhecido como *Intelligent Agents* trás o conceito de agentes para uma vertente mais comercial de software. É a terceira geração da plataforma de agentes que reune um conjunto de componentes com elevada eficácia numa estrutura modular bastante completa [HRHL01].
JAM	JAM é uma arquitetura de agentes inteligentes hibrida que integra um Sistema de Raciocínio Procedimental, Semântica de Circuitos Estruturados e Planos. Permite o raciocínio sobre vários objetivos em simultâneo, é orientado aos objetivos e a eventos. Ao nível das funcionalidades ainda tem algumas lacunas no que diz respeito a geração de planos e aprendizagem [Hub99].
Jason	Jason é um interpretador para uma versão baseada no AgentSpeak. Implementa uma plataforma para o desenvolvimento de agentes com um operador semântico da linguagem do próprio AgentSpeak. Pode ainda tirar partido utilizado em conjunto com o JADE propagando os agentes pela rede. Utiliza planos com linguagem de alto nível e permite algum nível de customização de determinados componentes [RHB07].

é totalmente grátis podendo ser descarregado do site oficial[1] na versão disponível mais recente, ou descarregar a versão desejada e caso seja necessário correr as atualizações automáticas para corrigir erros e atualizar as bibliotecas incluídas.

Para a instalação foi descarregado o jME3 SDK para versão Windows de 32bit. Esta plataforma consome poucos recursos e só necessita de 40MB de memória para a JVM, mais o espaço dos modelos gráficos. Não necessita de muito processamento pelo que corre em máquinas com requisitos mínimos sem problemas. Para além do Software Development Kit (SDK) do jME3 só é necessário ter instalado o *Java Development Kit* (JDK) ou utilizar o compatível instalado automaticamente.

Este pode ser utilizado em qualquer plataforma com mínimas alterações e o seu funcionamento será exatamente igual nos sistemas operativos mais conhecidos. Pode ainda ser também adaptado para ser utilizado em plataformas móveis com alguns ajustes uma vez que o Java também se encontra disponível nestes dispositivos.

Está construído sobre uma arquitetura modular, pelo que se podem instalar vários módulos

[1] `http://jmonkeyengine.org/`

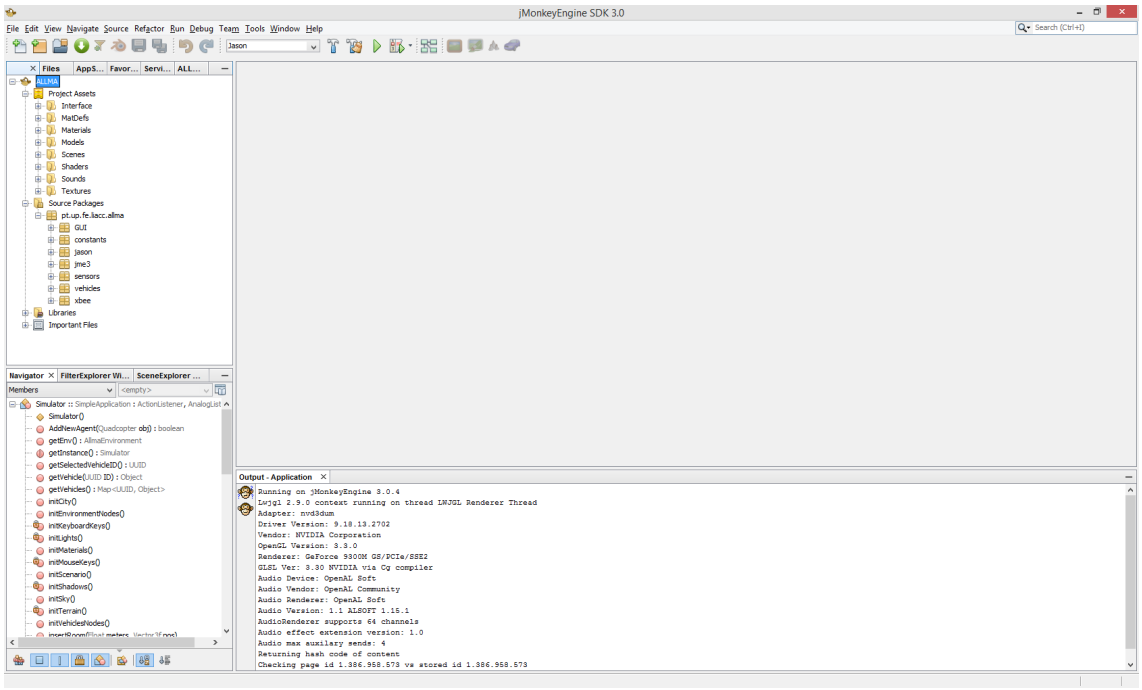


Figura A.1: jMonkeyEngine SDK 3.0

adicionais dos repositórios oficiais, resultando num ambiente de desenvolvimento bastante completo. Ao nível das funcionalidades inclui um editor de modelos integrado e permite a utilização de outros externos através de módulos externos. Tem acesso às ferramentas de interfaces SWIG pelo que se consegue desenvolver as interfaces GUI para além do ambiente virtual 3D. Inclui o motor de simulação físico *Bullet* para tornar os ambientes 3D mais realistas e ainda permite suportar vários formatos de ficheiros.

Outras bibliotecas que sejam necessárias podem ser adicionadas diretamente ao projeto ou se necessário disponibilizá-las para todos os projetos em simultâneo.

A.2.2 Ambiente de Desenvolvimento de Sistemas Multi-Agente

Para o desenvolvimento do SMA foi selecionado o Jason que implementa uma arquitetura BDI. Esta é uma extensão da linguagem de programação abstrata AgentSpeak criada por Anand Rao e encontra-se implementado na linguagem de programação Java. Jason é considerado um interpretador de planos de linguagem de alto nível para uma versão melhorada do próprio AgentSpeak. Integra muitas outras funcionalidades como pressupostos de mundo fechado e mundo aberto, tratamento de planos em caso de falha, comunicação entre agentes através de *speech-act*.

O ambiente virtual é implementado em Java e não em AgentSpeak e dá suporte para a utilização de agentes distribuídos pela rede usando o JADE, e este ainda permite a agregação de outras infraestruturas. Pode ser bastante adaptável, podendo indicar uma arquitetura para cada tipo de agente, no que diz respeito às suas funções, perceções, crenças e comunicação entre agentes. As

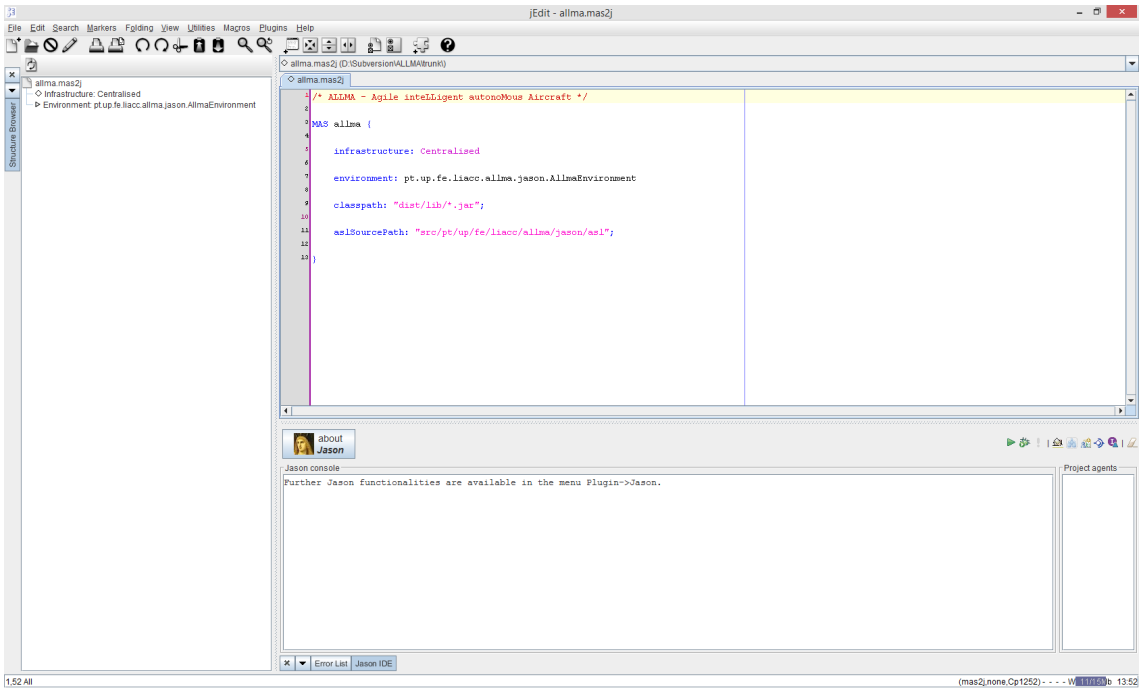


Figura A.2: jEdit

suas ações são executadas através de *internal actions* que permite a interface entre o Jason e o Java.

É um programa de código-fonte livre e é distribuído sobre a licença GNU LGPL podendo ser descarregado do site oficial[2] na versão mais recente disponível. Para a instalação foi descarregado o Jason-1.4.0a. Esta plataforma consome poucos recursos e não necessita de instalação prévia, bastando apenas ser descomprimido o ficheiro para o diretório desejado. Conta ainda com um módulo disponibilizado para o Eclipse ou um IDE, o jEdit que permite executar o *mind inspector*, ou seja, a depuração dos agentes.

Por fim falta referir como executar projetos Jason diretamente no executável da interface gráfica, para tal é necessário executar uma configuração nas propriedades do projeto do jME3. Especificar nas propriedades da aplicação a *main.Class* como *jason.infra.centralised.RunCentralisedMAS* com os argumentos *allma.mas2j*.

A.2.3 Ambiente de Configuração dos módulos XBee

Por forma a se poder utilizar a Java XBee API, os módulos de rádio devem então ser configurados no modo API, permitindo o envio e receção dos pacotes e normalmente basta configurar o parâmetro AP como 2. Os pacotes que podem ser processados dependem do tipo da série utilizada, no caso da integração presente foram utilizados os modelos da série 2 com a respetiva última atualização disponível do *firmware*.

[2] `http://jason.sourceforge.net/wp/`

No entanto se os módulos nunca foram configurados é necessário proceder à instalação do *firmware* uma vez que vêm de fábrica configurados com o *firmware* AT, por exemplo, *coordinator*, *router* ou *endpoint*. Para se executar este procedimento é necessário utilizar a ferramenta X-CTU[3] disponível para Windows.

O procedimento da configuração numa primeira abordagem com estes módulos pode ser um pouco difícil de entender, mas é um processo bastante fácil e rápido de se executar. Como já foi referido é necessário descarregar a ferramenta X-CTU que permite facilitar a configuração dos módulos com as configurações que forem necessárias e ter dois módulos XBee serie 2 e o respetivo adaptador USB.

Uma vez que esteja instalado é preciso definir como vai ser estruturada a rede, de acordo com a arquitetura da plataforma deve-se configurar o XBee que se encontrar na plataforma de simulação como Router API e para cada VANT que se necessite de conectar à rede deve ser configurado como *Coordinator API* ou *Coordinator AT*.

Todas as redes XBee devem ter um ou mais *Coordinators* que fazem a gestão da rede e neste caso só é necessário um *Router* visto ser um dispositivo que reúne os pacotes e faz o seu tratamento.

Para começar a configuração basta ligar o módulo, selecionar a porta série e velocidade correta no separador das definições no X-CTU como podem verificar na Figura A.3. Neste separador são listadas todas as portas série, a porta do adaptador USB é representada como uma porta serie virtual.

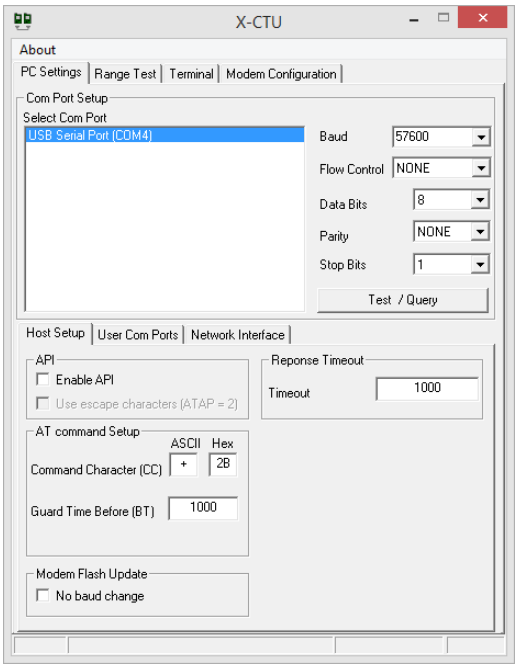


Figura A.3: X-CTU - Definições

[3] http://www.digi.com/support/productdetail?pid=3352&osvid=57&type=utilities

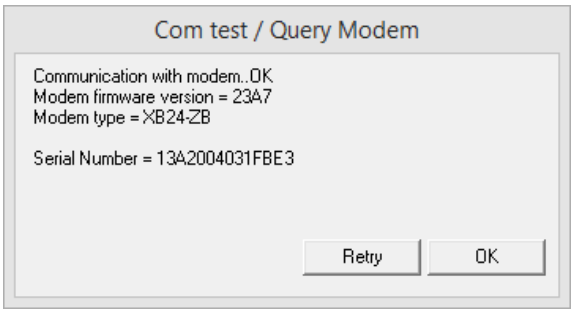


Figura A.4: X-CTU - Teste do Módulo XBee

Para se verificar que o XBee se encontra a comunicar corretamente deve-se pressionar o botão teste, o X-CTU vai tentar estabelecer a comunicação com o módulo XBee e se obtiver sucesso, responderá com dados sobre o *firmware* instalado como é apresentado na Figura A.4.

Caso o X-CTU não encontre o módulo provavelmente se encontra danificado ou as opções de comunicação não estão selecionadas corretamente, podendo tentar usar o botão de *reset* para tentar obter acesso e reprogramar o firmware de novo com todas as configurações necessárias.

Os módulos XBee oferecem muitas funcionalidades e por isso têm muitas configurações que podem ser alteradas. Nesta configuração só precisamos de definir e anotar três parâmetros essenciais para se criar a rede de comunicação desejada que são o identificador de rede onde se vão trocar as mensagens, e os endereços dos módulos. Todas as mensagens que não se identificarem dentro destes parâmetros são automaticamente descartadas.

Para efetuar este procedimento deve-se escolher o separador da configuração do módulo no X-CTU, onde se encontram todos os parâmetros, *firmwares* e respetivas versões disponíveis para a configuração do módulo, como se pode verificar na Figura A.5.

Uma vez que o *firmware Router API* esteja programado, deve-se então fazer a leitura das suas definições através do botão de leitura para verificar se é necessário escolher outro identificador da rede local privada (PAN) e caso seja necessário alterar algum, deve-se gravar essas alterações de seguida.

Para além disso é necessário anotar os endereços de destino do módulo, ou seja, é necessário conhecer o número de série para serem configurados nos restantes módulos dos VANTs, estes números encontram-se divididos em dois tipos como por exemplo SH:0013A200 e SL:4031FBE3.

Este procedimento foi efetuado para a configuração do módulo da plataforma de simulação e para a configuração dos restantes módulos basta efetuar o mesmo procedimento selecionando o *firmware* para a versão *Coordinator* e configurar os restantes parâmetros com os dados anotados.

A.2.4 Estação de Controlo no Solo

Para a comunicação com o sistema físico e receção dos dados de telemetria foi utilizado o programa oficial OpenPilot GCS. Para além da sua funcionalidade principal, que é configurar o *hardware* da placa OpenPilot CC3D também permite aceder à telemetria deste. É um programa de

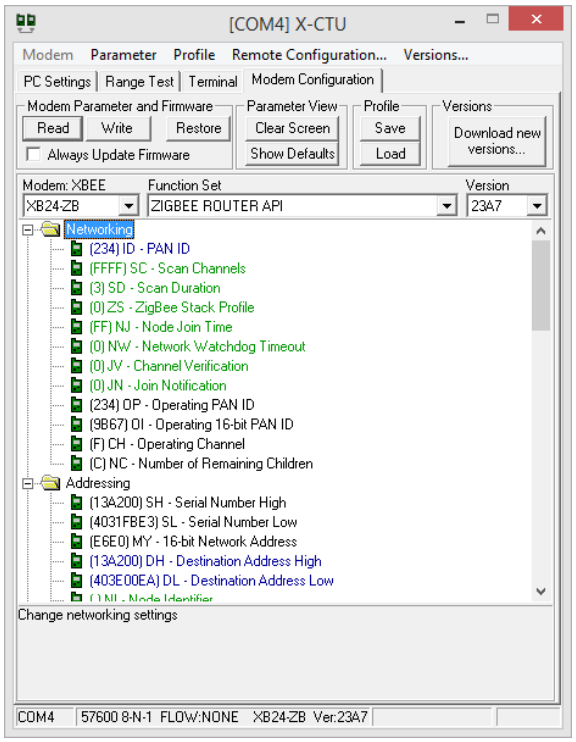


Figura A.5: X-CTU - Configuração do Módulo XBee

código-fonte livre e é totalmente grátis, onde toda a sua comunidade contribui para o seu desenvolvimento. Pode ser descarregado do site oficial[4] na versão mais recente disponível. É boa prática utilizar sempre a versão estável dado que muitas das vezes são introduzidos erros de códigos nas versões em desenvolvimento.

Para a instalação foi descarregado o OpenPilot-RELEASE-13.06.04 para versão Windows de 32bit, também se encontra disponível para Mac e Linux. Este programa consome poucos recursos e não necessita de instalação prévia. Como já foi referido, o ECS pode servir de interface do VANT uma vez que tem a capacidade de receber a telemetria, sendo assim consegue enviar pacotes com as informações dos sensores.

A telemetria é estruturada, monitorizada e controlada por três componentes principais: o *UAVObjectManager* que armazena todos os dados, o UAVTalk que é o protocolo de comunicação e o interpretador de pacotes. Estes componentes vão garantir que os dados se encontram sincronizados no ECS e no VANT.

A.3 Resultado da Instalação

Após a instalação, atualização e configuração de todas as ferramentas, encontra-se operacional todo o ambiente de desenvolvimento necessário para a implementação da plataforma de simulação. Na Figura A.1 apresenta-se o ambiente de desenvolvimento do *jMonkeyEngine 3.0* em modo

[4] `http://www.openpilot.org/`

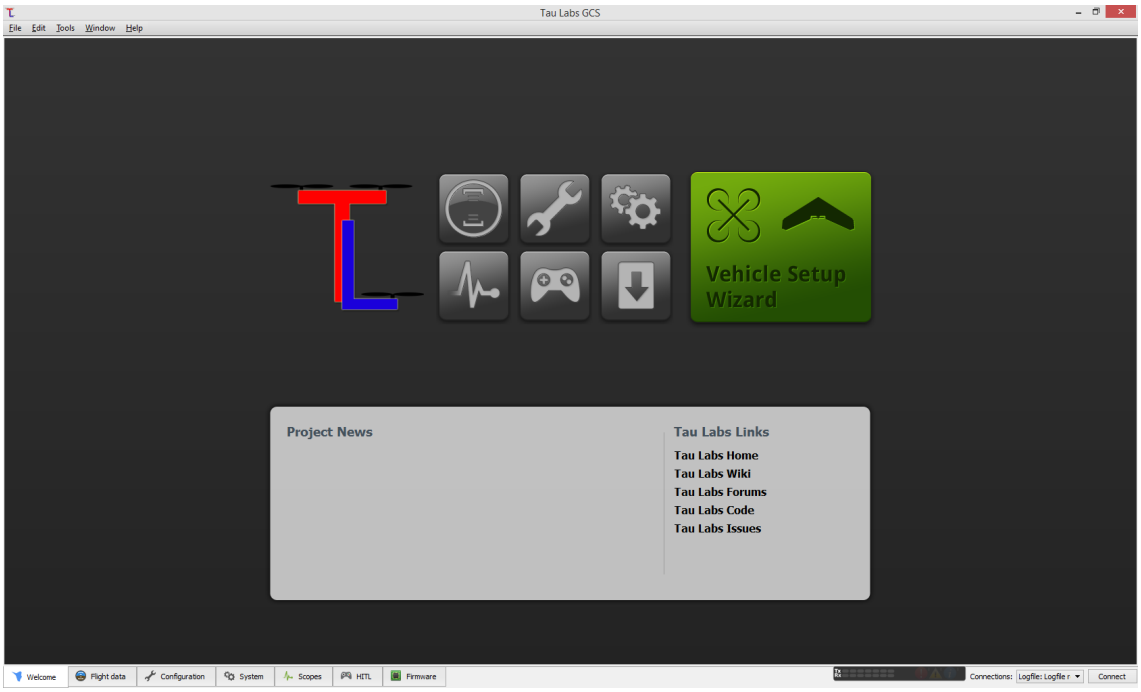


Figura A.6: Tau Labs GCS

de execução. Na Figura A.2 apresenta-se o ambiente de desenvolvimento *jEdit* para os agentes em Jason em modo de execução e na Figura A.3 é apresentada a ferramenta de configuração dos módulos XBee. Por último na Figura A.6 apresenta-se o *Tau Labs GCS*, uma alternativa compatível ao OpenPilot GCS para o ambiente de comunicação com o equipamento real.

A.4 Conclusão

O processo de instalação no ambiente de desenvolvimento Windows é uma tarefa relativamente simples e pouco demorada, uma vez que todas as ferramentas apresentam configuradores automáticos. Na falta destes configuradores, significa que só é necessário descomprimir o conteúdo do respetivo ficheiro e procurar o executável.

A adoção do ambiente Windows para o desenvolvimento não significa que esta mesma solução não esteja presente nos restantes sistemas operativos. Outra solução pode passar pela utilização de alguma distribuição do Linux, especificamente Ubuntu, no qual podem descarregar os ficheiros das páginas oficiais e proceder à instalação da mesma forma. A única diferença é que poderá ser necessário compilar as ferramentas antes de serem executadas. Por exemplo o Jason já inclui um *script* que executa todo o procedimento automaticamente através da instrução como "run bin/jason.sh".

Após o estudo das tecnologias, o Java foi escolhido como linguagem principal e as tecnologias selecionadas são baseadas em Java o que permite desde logo a interoperabilidade de todos os

componentes do projeto. O Java foi escolhido pela capacidade de poder ser executado nos sistemas operativos mais utilizados e ainda de se poder facilmente adaptar às plataformas móveis.

Em relação às ferramentas selecionadas, o jME3 foi escolhido por ser baseado em Java, implementar um motor 3D bastante completo e ser simples de utilizar. Incluindo também várias funcionalidades que correspondem aos requisitos necessários para o projeto. O mesmo se pode referir do Jason, este também é baseado em Java e integra a arquitetura BDI pretendida. As suas funcionalidades e o facto de implementar os agentes através da implementação de planos com um alto nível de abstração foi também um dos fatores que pesou na decisão.

Apêndice B

Artigo

Neste apêndice encontra-se uma referência ao artigo que foi aceite para submissão e para apresentação em conferência. Fazendo uma breve descrição da conferência e de seguida a apresentação do resumo do respetivo artigo.

B.1 SIMUTools 2014

SIMUTools 2014 é a sétima edição da Conferência Internacional anual sobre Ferramentas de Simulação e Técnicas. A conferência centra-se num vasto conjunto de desafios de investigação no campo da simulação, modelagem e análise, abordando as tendências atuais e futuras de técnicas de simulação, de modelos, de práticas e de *software*. A conferência é dedicada a promover a investigação em colaboração interdisciplinar nessas áreas e num amplo espectro de domínios de aplicação[1].

B.1.1 *A Platform for the Design, Simulation and Development of Quadcopter Multi-Agent Systems*

Os veículos aéreos não tripulados (VANTs) estão a ganhar a atenção dos investigadores em todo o mundo, devido à sua capacidade de manobra e desempenho tanto em ambientes interiores como exteriores. O objetivo do projeto descrito neste artigo é especificar e implementar uma plataforma de simulação simbiótica para auxiliar na conceção, desenvolvimento e teste dos VANTs.

Neste artigo, primeiro é analisada a integração de duas ferramentas de desenvolvimento para construção do ambiente de simulação. A plataforma Jason, sendo utilizada para o desenvolvimento e simulação de sistemas multiagente e a plataforma jMonkeyEngine 3.0 para a implementação do ambiente gráfico. De seguida descreve-se a implementação do sistema físico do VANT.

Este artigo apresenta resultados preliminares sobre a descrição da plataforma de simulação simbiótica.

[1] `http://simutools.org/2014/`

Apêndice C

UML

Neste apêndice encontram-se os diagramas Unified Modeling Language (UML) mais significativos da plataforma de simulação.

C.1 Diagrama de Camadas

C.2 Diagramas de Casos de Uso

C.3 Diagrama de Classes

C.4 Diagrama de Fluxo de Dados

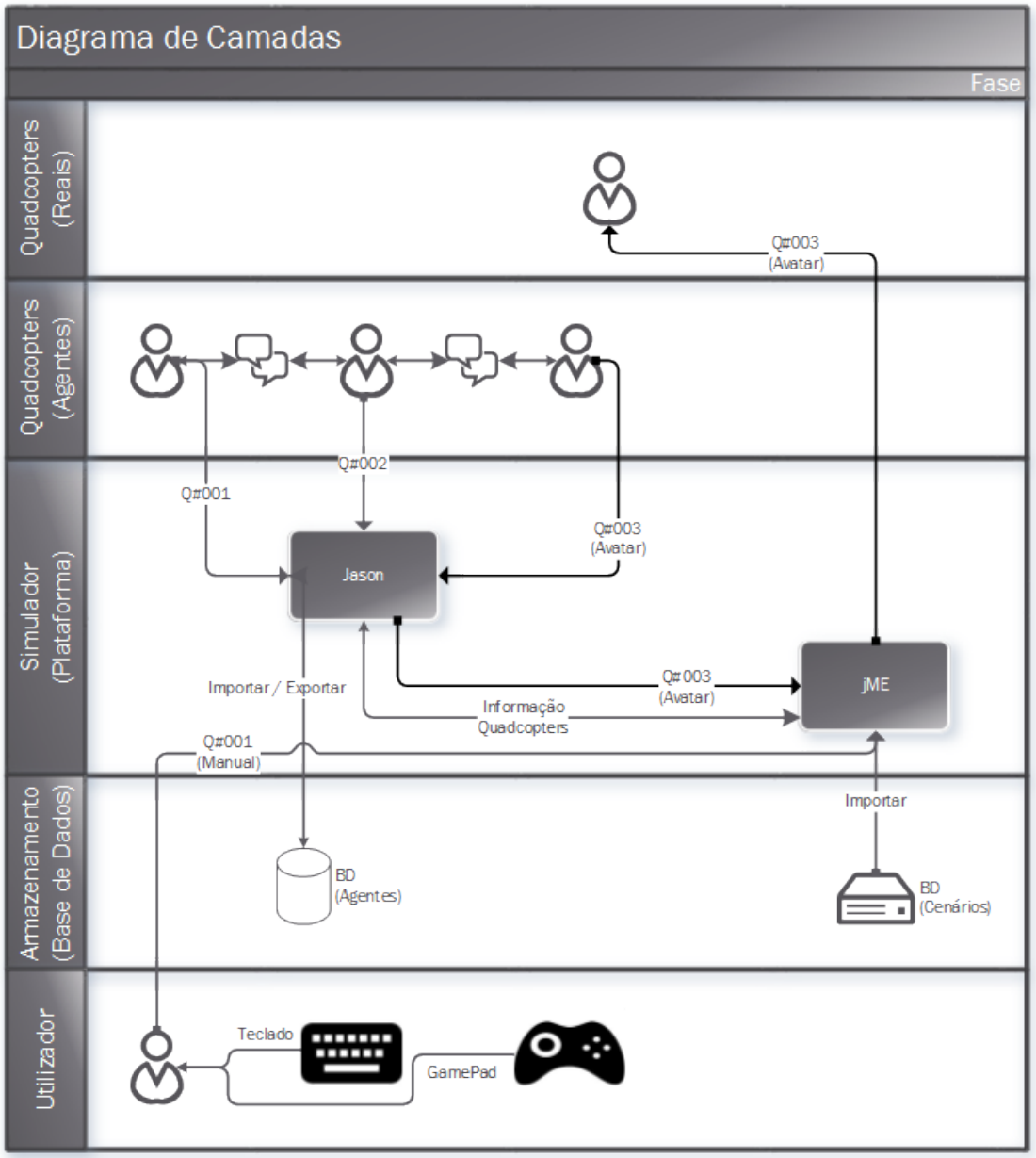


Figura C.1: Diagrama de camadas: Plataforma

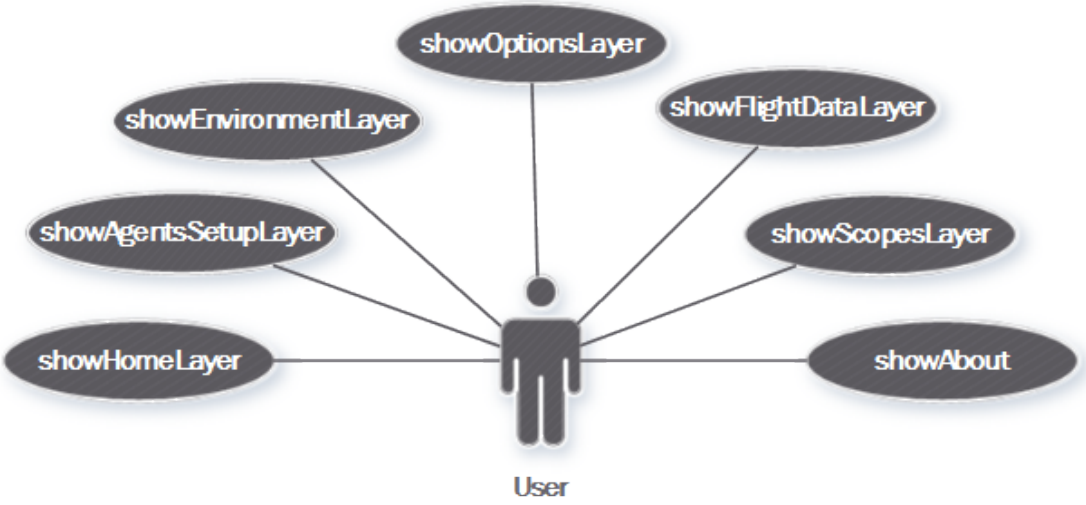


Figura C.2: Diagrama de caso de uso: *InterfaceLayer*

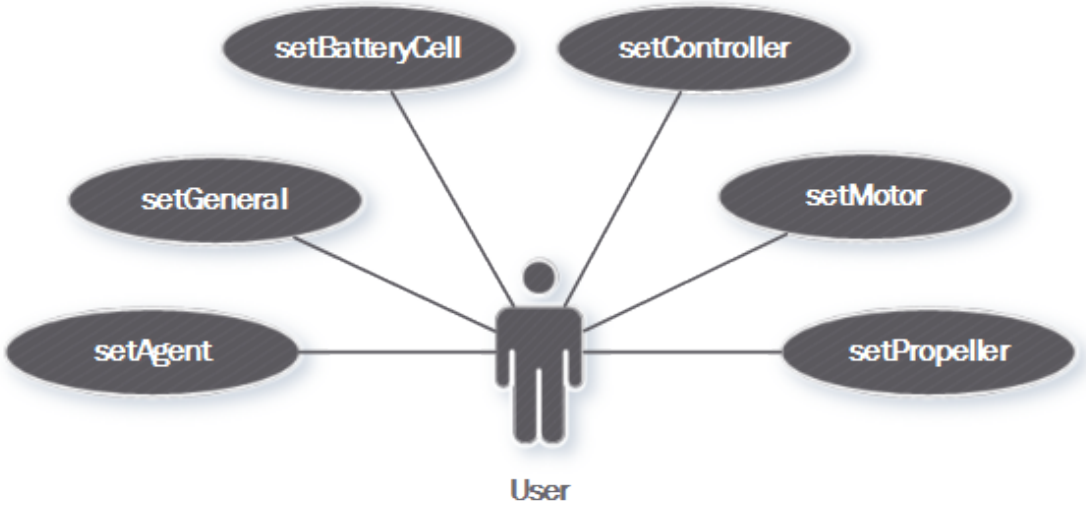


Figura C.3: Diagrama de caso de uso: *AgentsSetupLayer*

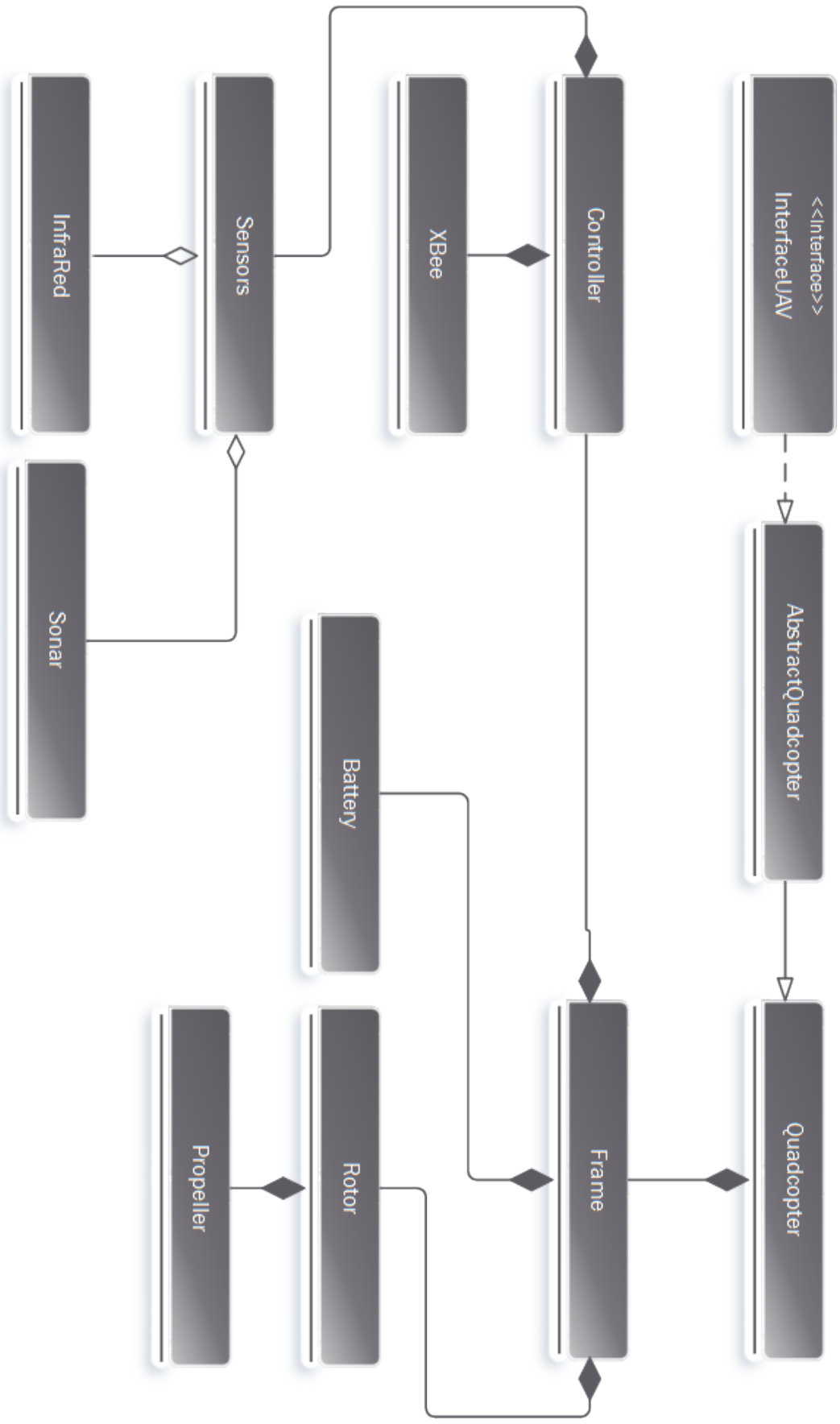


Figura C.4: Diagrama de classes: *Structure*

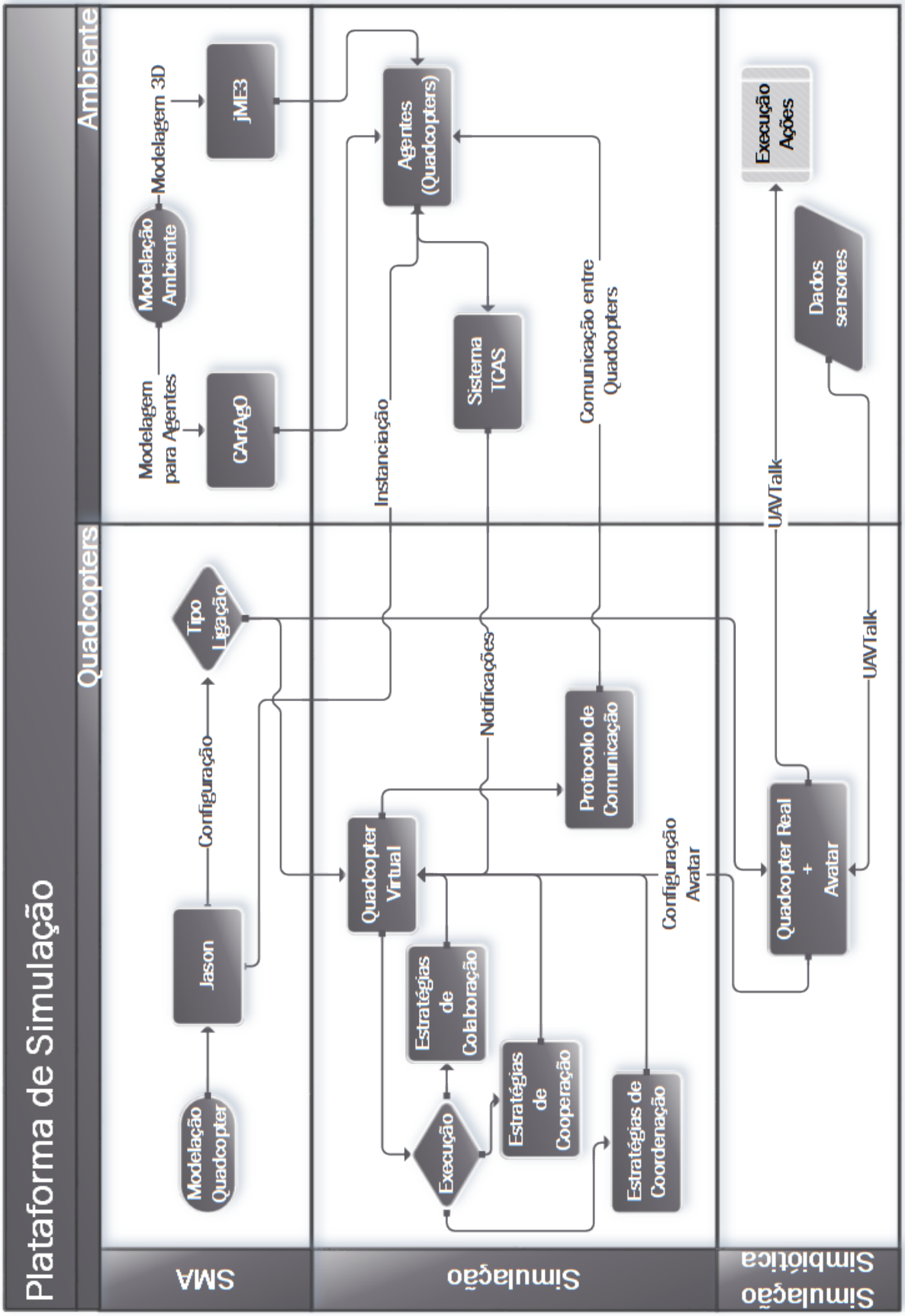


Figura C.5: Diagrama de Fluxo de Dados: Componentes da Plataforma